Manual Usuario gvHidra

Versión 1.0.0

Tabla de contenidos

Lista de tablas

Parte I. Empezando con gvHidra

Tabla de contenidos

Capítulo 1. Empezando con gvHIDRA

1.1. Introducción

1.1.1. Acerca de este documento

Este documento pretende ser un punto de referencia para todo aquel que se vaya a embarcar en el desarrollo de aplicaciones basadas en el *framework* gvHidra.

A lo largo de los diferentes capítulos, se van a ir mostrando las posibilidades del *framework*, las mejoras que aporta frente a otras opciones y, por supuesto, como instalar y ejecutar una primera aplicación basica. Con todo ello, se tendrá preparado un entorno de trabajo funcional para poder ir desarrollando y practicando los conceptos adquiridos a lo largo de este manual.

Este manual ha sido desarrollado por el equipo de gvHIDRA para favorecer y agilizar el acceso a cualquier usuario novel o avanzado en el desarrollo de aplicaciones basadas en este mismo *framework*.

Complementariamente existe una lista de distribución [http://listserv.gva.es/cgi-bin/mailman/listinfo/gvhidra_soporte] a la que cualquier desarrollador puede suscribirse y hacer llegar sus peticiones de ayuda o sugerencias para mejorar la herramienta.

Por otro lado existen unos servicios de archivo que permiten realizar búsquedas en el histórico de mensajes de la lista simulando el funcionamiento tipo "foro". Se puede acceder a las listas de gvHIDRA desde la interfaz de nabble [http://gvhidra.3754916.n2.nabble.com], obtener orígenes RSS e incluso escribir a la lista.

1.1.2. ¿Qué es gvHIDRA?

gvHidra son las iniciales de **G**eneralitat **V**alenciana: **H**erramienta **I**ntegral de **D**esarrollo **R**ápido de **A**plicaciones.

Si hay que dar una explicación rápida de qué es este proyecto, podemos decir que se trata de un entorno de trabajo (*framework*) para el desarrollo de aplicaciones de gestión en entornos web con PHP siguiendo una guía de estilo (una guía para unificar los criterios de aspecto y usabilidad en el proceso de desarrollo de aplicaciones).

Evidentemente, esto no responde a la pregunta que da título a este punto. Por ello, vamos a explicar las motivaciones que nos llevaron a su creación y las características que cubre.

1.1.2.1. Un poco de historia

El Servicio de Organización e Informática (SOI) de la Conselleria de Infraestructuras y Transporte de la Generalitat Valenciana (CIT) ha trabajado tradicionalmente con la premisa de aumentar la productividad en base a entornos de trabajo que facilitan el desarrollo de aplicaciones. Concretamente, una de sus líneas de desarrollo, utilizaba unas plantillas en PowerBuilder que, en una arquitectura cliente servidor, reducían los tiempos de desarrollos resolviendo la parte general del problema.

Dentro de este marco, la CIT emprendió el proyecto gvPONTIS cuyo objetivo principal era migrar todos los sistemas a sistemas de código abierto. Entre otras facetas afectadas se encontraban los lenguajes de programación (se seleccionaron PHP y Java), los SGBD (se decidió fomentar el uso de Postgresql), la arquitectura (pasar al desarrollo web/tres capas)...

Con todo ello, se decidió crear un proyecto en PHP que, basándose en la guía de estilo de las aplicaciones de la CIT, aportara las mismas ventajas que las anteriores plantillas de PowerBuilder: aumentar la productiva de nuestros desarrollos.

Pero claro, a todos esos requerimientos, teníamos que añadir las dificultades que generaba el nuevo ámbito de trabajo: el entorno web (HTML, Javacript,...), el enfoque OpenSource, ... Por tanto se decidió incorporar como requerimiento la simplificación del entorno de trabajo para un desarrollador.

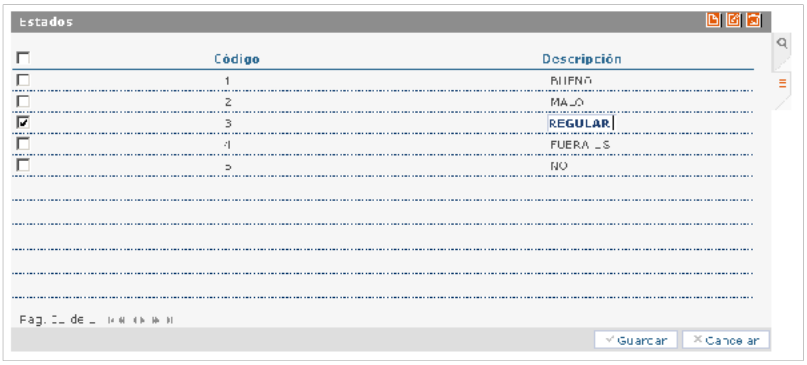
Con todo ello se creó un proyecto (igep: Implementación de la Guía de Estilo en Php) que componía el core del framework cuya primera versión estable salió el 16-11-2004. Este proyecto, al ser liberado con licencia GPL tomó la denominación gvHIDRA. A partir de este hito, empezaron a colaborar con el proyecto numerosas entidades públicas y privadas que nos ayudan a mantener el proyecto vivo y actualizado.

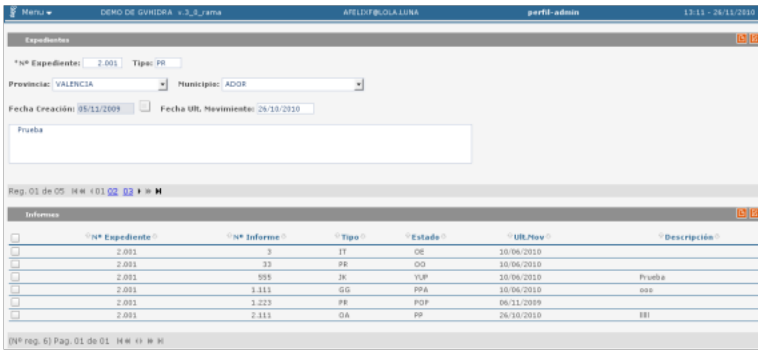
1.1.2.2. Pero ¿Qué es gvHIDRA?

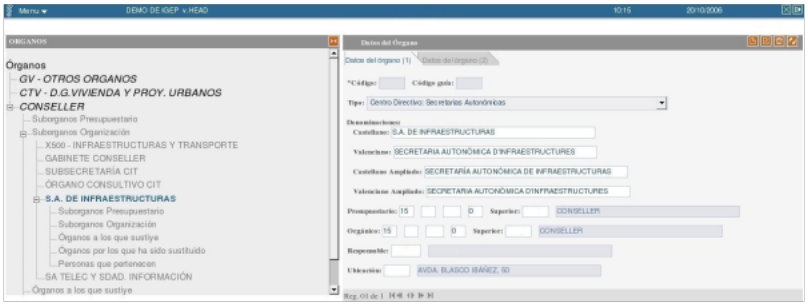
Con lo expuesto en el punto anterior, quedan claras las premisas que nos movieron a plantearnos la creación de este proyecto pero, ¿Cómo hemos logrado unos objetivos tan ambiciosos? Bien, es dificil afirmar que los hemos alcanzado todos, pero creemos que buena parte de ellos quedan resueltos en el proyecto con las siguientes características del *framework*:

- **Patrones de interfaz**

  Una de las tareas más costosas es en la definición de la interfaz con el usuario. Para facilitar el trabajo, se han definido una serie de patrones de básicos. Estos patrones definen la forma de representación de la información (formato tabular, registro,...) y la forma con la que interacciona el usuario con dicha información. Esto nos lleva a que con la selección de un patrón obtenemos el diseño global de la ventana.



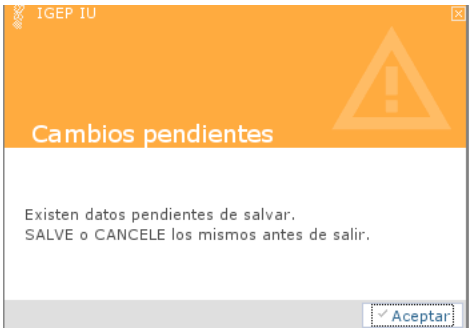




  Ejemplos de patrón simple tabular, maestro detalle y árbol.

- **Componentes complejos**

  La experiencia acumulada nos dice que todas las aplicaciones necesitan de una serie de componentes “complejos”. Ventanas de selección (en PowerBuilder listas de valores), listas enlazadas, acciones de interfaz (en WEB tec. AJAX), mensajes de información,... El *framework* genera estos componentes simplificando su utilización en las aplicaciones.



- **Operaciones preprogramadas y parametrizables**

  Al igual que con los componentes, hemos advertido que cierta problemática se repite en todas las aplicaciones que desarrollamos. Por esa razón, la hemos generalizado y resuelto en el *framework*, siendo incorporada a las aplicaciones de forma transparente. Algunos ejemplos son:

  - Control de acceso concurrente: es importante que el garantizar la integridad de los cambios realizados por ello el *framework* incorpora de forma transparente un mecanismo de control.
  - CRUD: el *framework* genera las sentencias SQL necesarias para Crear, Leer, Actualizar y Borrar un registro.
  - Persistencia y validación de tipos: completando lo que nos ofrece el PHP, el *framework* incorpora objectos persistentes y validación de tipos de datos.

- **Soporte a diferentes SGBD**

  A través del proyecto PEAR::MDB2, el *framework* permite trabajar con diversos SGBD. Además, incorpora un capa de intermedia propia del *framework* que nos permite independizarnos de las diferentes interpretaciones del SQL que hace cada gestor (definición de límites, transacciones, ...).

- **Listados e informes**

  Uno de los mayores retos que hemos encontrado en el ámbito del proyecto fue generar informes de forma tan versatil como los datawindows de PowerBuilder. Gracias al proyecto jasperreports lo hemos conseguido. Apoyados por herramientas como el iReport, conseguimos listados e informes muy completos y en diferentes formatos.

- **Arquitectura MVC**

  El *framework* garantiza la arquitectura MVC en todos nuestros desarrollos forzando la separación de la lógica de negocio de la presentación mediante la distribución física de los ficheros fija.

- **Control de la vista**

  Como hemos comentado, uno de los objectivos principales de la herramienta es simplificar la labor del programador. Con esta premisa, hemos conseguido que un desarrollador de gvHIDRA realice una aplicación WEB sin necesidad de introducir ninguna línea de HTML o Javascript. La idea es que centre su trabajo únicamente en PHP, siendo así mucho más productivo.

- **Custom y temas**

  La herramienta está pensada para su despliegue en diferentes organizaciones, por ello, se distribuye en una arquitectura App/Custom/Core que permite modificar tanto el aspectos (CSS, imágenes, ...) como definir comportamientos propios de la organización.

- **Testing**

Incorpora la herramienta PHPUnit para que se puedan realizar testeos sobre el código generado. También se incorporan consejos y reglas para poder realizar test automáticos son Selenium.

- **Autentificación, auditoría y depuración**

  Para poder incorporarse en diferentes organismos incorpora un mecanismo de validación extensible siendo capaz de acloplarse a cualquier sistema de validación a través de PHP. Dispone de herramientas para reliazar auditorías y depuración.

Como buen proyecto opensource, el proyecto siguen en constante evolución incorporando nuevas funcionalidades que nos permitan, ante todo, ser más productivos.

1.1.3. Versiones

Desde su liberación, el xx-xx-2xxx, gvHIDRA ha seguido una estrategia de versiones con el fin de incorporar nuevas funcionalidades y resolver errores siendo la versión actual la 3.1.0 (rama 3.1.x). La numeración de las versiones corresponde al siguiente patrón: x.y.z Un cambio en el último número (z) implica corrección de errores/bugs, en este caso será invisible el cambio de versión en lo desarrollado a partir del framework. Si el cambio es en el segundo número (y), implica una mejora del framework o la adición de una nueva funcionalidad, esto sí puede provocar algún cambio en la programación de lo desarrollado a partir del framework. Por último, el cambio del primer número (x) sí que lleva una gran mejora en el framework o una nueva funcionalidad importante.

Recomendaciones:

- Trabajar siempre en la versión más reciente posible, para facilitar la resolución de errores, evitar usar una versión correspondiente a una rama no activa. Esto no significa que se descarten problemas de versiones anteriores, sino que si la solución implica cambios, éstos no se publicarán en ramas no activas.

1.2. Requerimientos

En este punto se explica las necesidades de un puesto cliente para poder ejecutar una aplicación realizada con gvHidra.

1.2.1. Hardware

Los requerimientos de hardware **minimos** recomendados para un correcto funcionamiento de una aplicación desarrollada con gvHidra son los siguientes:

- Pentium IV 2GHz o superior
- RAM 256 MB o superior

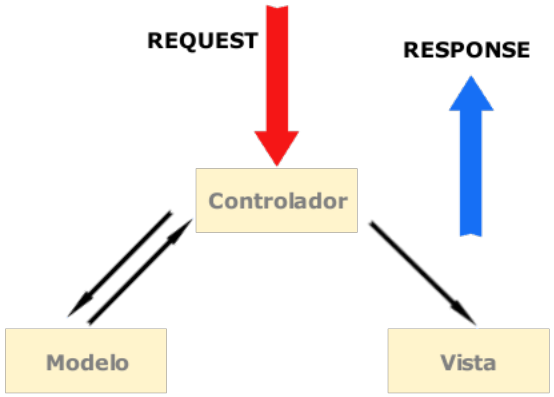
1.2.2. Software

En el caso de los requerimientos de software **minimos**

- Navegador mínimo: Firefox 3 o superior (actualizado el 17-01-2011) El navegador ha de permitir ventanas emergentes para la visualización de las ventanas de selección.
- Acrobat Reader (para visualizar listados)

1.3. Entendiendo el entorno

El *framework* gvHIDRA sigue una arquitectura MVC. Esta arquitectura tiene como objeto dividir en capas diferentes la lógica de negocio, la lógica de control y la presentación. Con ello conseguimos desarrollos más robustos y fuertes ante los cambios de requisitos. El siguiente diagrama muestra como funciona de forma esquemática la relación entre capas:



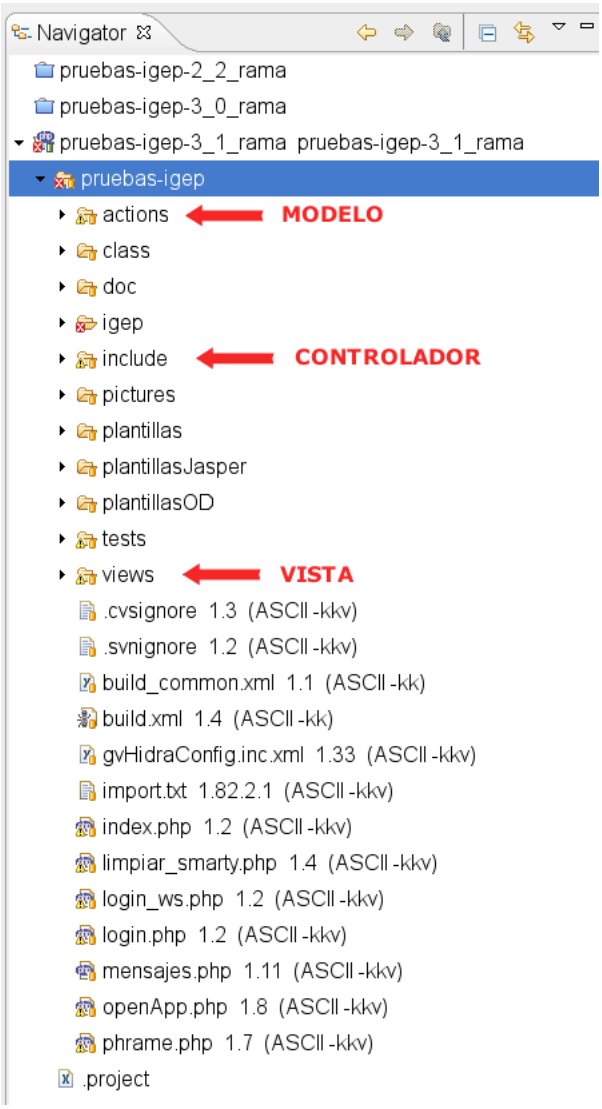
Esta división en capas se plasma físicamente esta división dentro de una aplicación gvHIDRA. Concretamente, para hacer un mantenimiento, el programador tendrá que trabajar en las siguientes partes de la estructura:

- **actions**: corresponde al modelo y en el se alojan las clases manejadoras. Estas clases nos permiten acceder a los datos y realizar los tratamientos sobre ellos. Muchos de los tratamientos y comportamientos vienen heredados de forma que nuestra clase simplemente debe añadir el comportamiento específico.

- **include/mappings.php**: corresponde con el controlador. Basado en el proyecto Phrame, se compone de un fichero que nos permite asociar una acción (solicitud del usuario) con la clase que lo va a resolver.

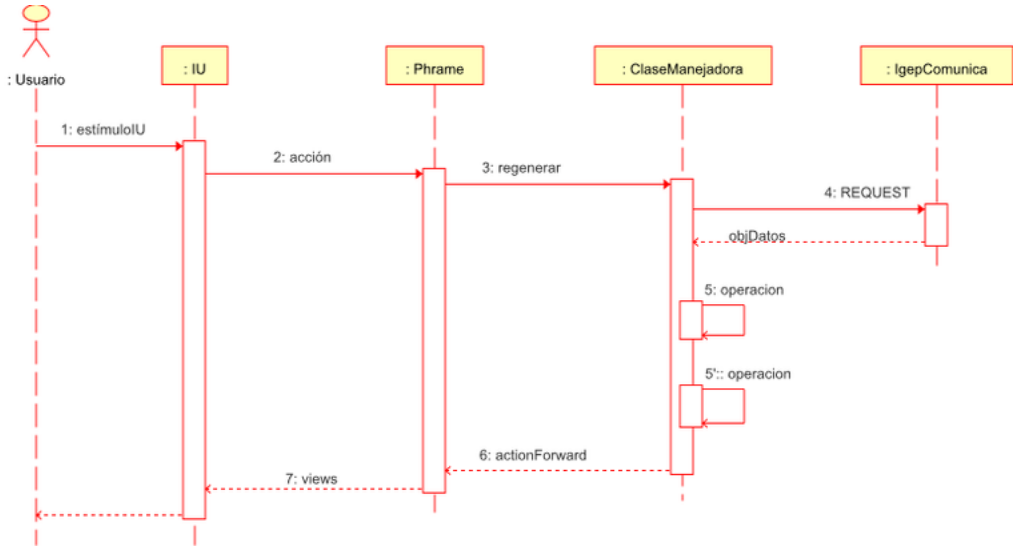
- **views**: corresponde con la vista. En este directorio encontraremos las vistas que designarán la pantalla a visualizarse.

Aquí tenemos un esquema donde lo podemos ver un ejemplo de aplicación con las capas que componen la arquitectura MVC.



A estas capas, falta añadir la interfaz con el usuario, las pantallas. Para la realización de las mismas, gvHIDRA hace uso de smarty y de unos plugins propios. Estos fichero, se ubican en el directorio templates y contienen la definición de una ventana con todos sus componentes.

Una vez vista la estructura física de una aplicación, es conveniente ver los componentes internos del framework a través del flujo interno que provoca una petición de pantalla. En el siguiente diagrama de secuencia, hemos colocado algunos de los actores que intervienen en el tratamiento de una petición así como sus operaciones.



1. El flujo se inicia con la aparición de un estímulo de pantalla lanzado por el usuario.
2. Este estímulo es transmitido al controler (en nuestro caso phrame) en forma de acción. Ahora es phrame quien, consultado con el fichero de mapeos (fichero mappings.php) es capaz de conocer que clase es la encargada de gestionar la acción. Una vez conocida la clase, la "levanta" y le cede el control pasándole todos los datos de la petición.
3. La clase (conocida como clase manejadora) una vez tiene el control realiza varios pasos.
   - Reconecta de forma automática a la base de datos (en el caso de que exista).
   - Parsea el contenido de la petición (REQUEST) encapsulando en un objeto iterador que agrupa el contenido en matrices por operación.
   - Lanza las distintas operaciones. Estas operaciones se extienden con el comportamiento extra que añade el programador. Es decir, si lanzamos una acción de borrado, el *framework* realiza las operaciones necesarias para eliminar la tupla de la base de datos y el usuario tiene dos puntos de extensión opcional de dicho comportamiento: antes de borrar (métodos pre: para validaciones) y después de borrar (métodos post: para operaciones encadenadas).
4. Una vez finalizado todo el proceso, la clase manejadora devuelve un actionForward a phrame. Este lo descompone y se localiza la views seleccionada.
5. Finalmente, el views recoge la información y, con la plantilla (fichero tpl de smarty) muestra la información en pantalla.

1.4. Instalación del entorno

Para ejecutar una aplicación en gvHIDRA, se tiene que disponer de un servidor web capaz de intepretar PHP. Estos son los requisitos técnicos:

- Servidor WEB. Puede ser cualquier servidor, aunque nosotros recomendamos Apache 2.X.
- PHP. Debe ser una versión de la rama 5.2.X. Actualmente, no garantizamos la compatibilidad con la versión 5.3.X aunque esperamos hacerlo en breve.
- PEAR. Para el correcto funcionamiento del framework, se requieren una serie de librerías del proyecto PEAR con sus dependencias respectivas. Las requeridas son:

- MDB2: capa de abstracción sobre la base de datos
- MDB2_Driver_x: correspondiente al SGBD con el que vamos a trabajar.
- PEAR: sistema básico de PEAR
- Auth: sistema de validación.
- Mail: para poder hacer uso de la clase de envio de correos del *framework*.
- SOAP: si se quiere hacer uso de servidores de web services. Aunque PHP incorpora la la extensión SOAP, la librería PEAR::SOAP ofrece mecanismos para genarar WSDL de forma sencilla.

1.4.1. ¿El entorno funciona correctamente?

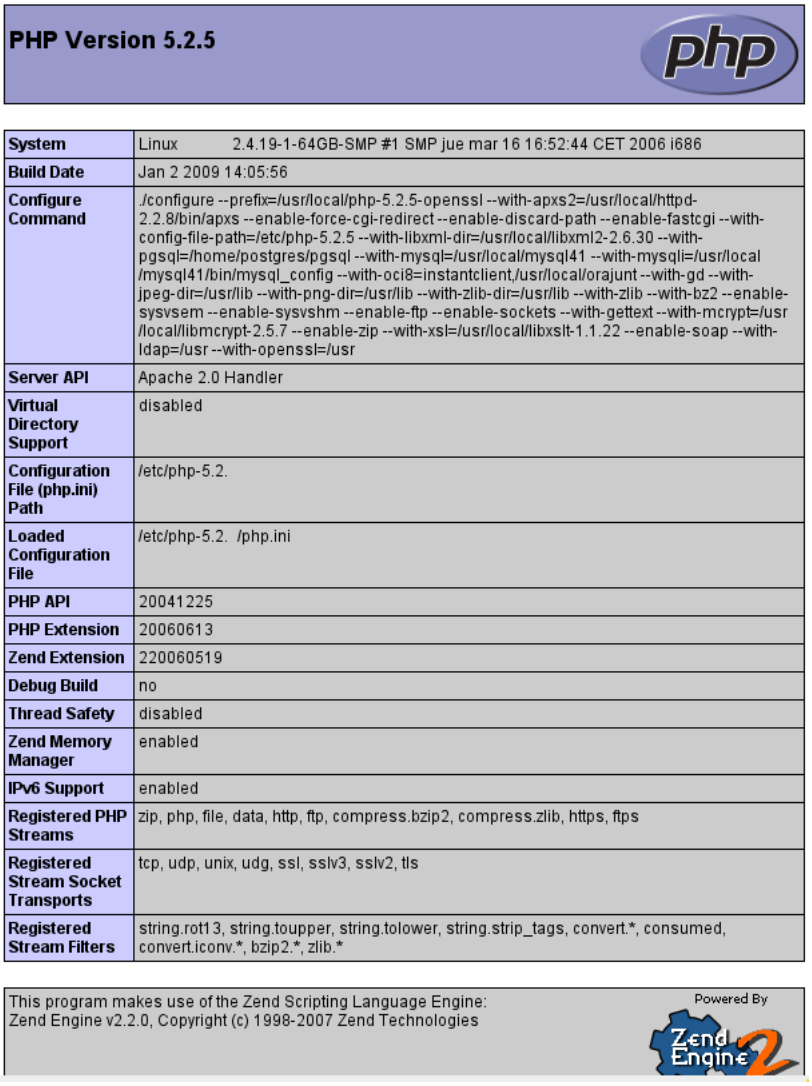
Lo primero que tenemos que comprobar es que el entorno de trabajo está funcionando correctamente. Para ello, vamos a ejecutar unos scripts que nos confirmarán que nuestro servidor está listo para albergar aplicaciones gvHIDRA.

1.4.1.1. ¿Funciona correctamente el PHP?

Creamos el siguiente script en la carpeta raiz del servidor web (generalmente, htdocs) con el nombre test1.php :

```
<?php
     phpinfo();
?>
```

Si todo ha ido bien, cuando accedamos a este script desde nuestro navegador tendremos algo parecido a la siguiente imagen.



PHP Version 5.2.5

System	Linux 2.4.19-1-64GB-SMP #1 SMP jue mar 16 16:52:44 CET 2006 i686
Build Date	Jan 2 2009 14:05:56
Configure Command	./configure --prefix=/usr/local/php-5.2.5-openssl --with-apxs2=/usr/local/httpd-2.2.8/bin/apxs --enable-force-cgi-redirect --enable-discard-path --enable-fastcgi --with-config-file-path=/etc/php-5.2.5 --with-libxml-dir=/usr/local/libxml2-2.6.30 --with-pgsql=/home/postgres/pgsql --with-mysql=/usr/local/mysql41 --with-mysqli=/usr/local/mysql41/bin/mysql_config --with-oci8=instantclient,/usr/local/orajunt --with-gd --with-jpeg-dir=/usr/lib --with-png-dir=/usr/lib --with-zlib-dir=/usr/lib --with-zlib --with-bz2 --enable-sysvsem --enable-sysvshm --enable-ftp --enable-sockets --with-gettext --with-mcrypt=/usr/local/libmcrypt-2.5.7 --enable-zip --with-xsl=/usr/local/libxslt-1.1.22 --enable-soap --with-ldap=/usr --with-openssl=/usr
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php-5.2.
Loaded Configuration File	/etc/php-5.2. /php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.2.0, Copyright (c) 1998-2007 Zend Technologies

Powered By Zend Engine 2

Podemos aprovechar para comproba que la versión de PHP es la correcta.

1.4.1.2. ¿Funciona la conexión al SGBD?

Bien, una vez hemos comprobado que tenemos una instalación correcta en lo relativo a PHP, vamos a comprobar que tenemos acceso a la base de datos. Para ello vamos crear un script que se conecte a nuestro SGBD. Para ello debemos copiar el siguiente código en el fichero test2.php:

```
<?php

include_once('MDB2.php');
include_once('PEAR.php');

$dsn= array(
  'phptype'  => 'xxxx',
  'username' => 'xxxx',
  'password' => 'xxxx',
  'hostspec' => 'xxxx',
  'port' => 'xxxx',
);

$options = array('portability' => MDB2_PORTABILITY_NONE,);
$res = MDB2::connect($dsn,$options);

if(PEAR::isError($res))
{
  print_r("Error:\n");
  print_r($res);
  die;
}
```

```
die("¡Eureka!");
?>
```

Ahora tenemos que completar la información con los datos necesarios para que localice el sistema localice la base de datos. Para ello, tenemos que completar la cadena de conexión con los siguientes valores:

- phptype: tipo de SGBD: oci8 (para Oracle), pgsql (PostgreSQL) o mysql.
- username: nombre de usuario para la conexión.
- password: password del usuario.
- hostspec: host del SGBD.
- database: nombre de la base de datos. No es necesario para Oracle.
- port: (opcional) puerto en el que está trabajando el SGBD. No es necesario si es el valor por defecto.

Con la información completa, sólo nos queda ubicar el fichero en una ubicación que sirva nuestro servidor web y acceder a la página con el navegador. Si todo ha funcionado correctamente debe aparecer en pantalla el texto "¡Eureka!". En caso contrario, conviene revisar los siguientes puntos:

- Instalación del paquete MDB2 y del driver MDB2 adecuado.
- Login y password del usuario.
- Acceso al SGBD, ¿tenemos acceso desde el servidor web a la BD?

1.5. Creando mi “hola mundo” en gvHidra

Una vez tenemos el entorno preparado para el *framework*, vamos a probar una aplicación de pruebas. Entramos en la web del proyecto [http://www.gvhidra.org] y descargamos la última versión.

En este paquete ZIP, tenemos los siguientes componentes:

- Directorio doc: aquí tenemos la documentación relativa a la versión en formato HTML.
- Directorio doxy: ficheros generados con el doxy-gen que facilitan una navegación rápida entre las funciones del framework.
- Directorio igep: core del framework. Es el directorio que deberemos copiar en todas nuestras aplicaciones para poder trabajar con gvHIDRA.
- Directorio Phpdoc: información generada por el phpdoc.
- Guía rápida.txt: pasos mínimos de prueba del paquete.
- License.txt: referencia a la licencia GPL.
- Readme.txt: información básica del proyecto.

Una vez descargado, seguiremos los estos pasos:

1. Obtenemos el archivo doc/plantilla-gvHidra.zip y lo descomprimimos en una carpeta del htdocs del servidor web.
2. Copiamos la carpeta igep que se encuentra en el paquete descargado en la carpeta creada en el paso anterior.
   Nota: puedes comprobar que la estructura es similar a la que hemos presentado en la imágen del punto 3.
3. Creamos carpeta templates_c y le damos permiso de escritura para el usuario Apache. Este directorio es el que utiliza el *framework* para compilar las plantillas.

4. Acceder a http://<servidor>/plantilla-gvHidra y validarse con el usuario 'invitado' y contraseña '1'.

Tras seguir estos pasos y, si todo ha ido bien, tenemos que obtener algo como esto:

Tras logarnos, entramos en la ventana principal de la aplicación que nos muestra las opciones de menu. En este ejemplo básico, sólo tenemos una opción correspondiente a un patrón tabular registro.

Al seleccionar la opción tabular-registro (donantes) entramos en una ventana que nos muestra dos tuplas. Este ejemplo trabaja sin conexión a base de datos (extiende la clase gvHidraForm_dummy), por lo que algunas de las funcionalidades no están habilitadas.

Enhorabuena ¡Ya tienes tu primera aplicación!

Capítulo 2. Conceptos técnicos de gvHidra

En este capítulo vamos a ver algunos conceptos teóricos que son necesarios antes de empezar a desarrollar aplicaciones con el framework. Con toda la información que comprende seremos capaces de entender e interpretar cada uno de los pasos que realicemos en capítulos posteriores.

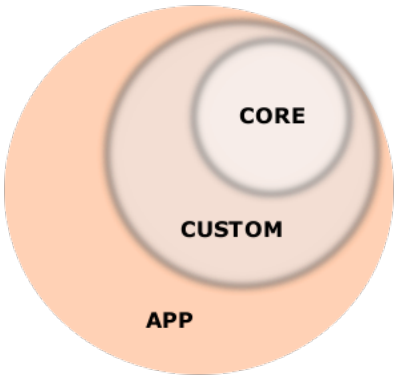
Se verán conceptos relativos a la arquitectura de las aplicaciones, a la interfaz con el usuario y a la lógica de negocio.

2.1. Arquitectura

gvHIDRA, como se ha comentado, es un proyecto opensource que se ha desarrollado para poder ser desplegado en entornos heterogeneos. Para ello su confirguración interna se ha organizado en una estructura de capas que permite redefinir los parámetros, crear clases específicas o cambiar el aspecto de cada una de nuestras aplicaciones.

2.1.1. Arquitectura por capas

Concretamente, el framework ofrece tres capas:



1. **core:** es la capa propia de los ficheros del framework. Está ubicada en el directorio igep y contiene ficheros de configuración propios del proyecto.
2. **custom:** destinado a las configuraciones propias de la entidad donde se despliega la aplicación. Generalmente, la organización donde se despliegue una aplicación tiene una serie de características propias:
   - aspecto: se puede configurar a partir de css e imágenes.
   - definición de listas o ventanas de selección generales (municipios, provincias, terceros, ...).
   - clases propias, tipos, ws, ...
   - configuraciones propias: conexiones a BBDD corporativas, acceso a datos comunes, formato de las fechas...
3. **app:** cada aplicación tiene una configuración propia:
   - acceso a la BBDD propios de la aplicación.
   - listas y ventanas de selección propias de la aplicación.
   - configuraciones propias de la aplicación: nombre aplicación, versión, modo de debug/auditoria, comportamiento de la búsqueda, ...

Resumiendo, debemos tener en cuenta que hay varios niveles dentro de la arquitectura de una aplicación gvHIDRA y que en aras de la reutilización, conviene definir un custom propio de la organización en la que nos encontramos.

2.1.2. Configuración

En cuanto a la carga de la configuración, el framework la realiza en dos fases consecutivas siguiendo en cada una de ellas el orden (core/custom/app). Las dos fases se corresponden con:

- **carga estática:** esta fase carga los parámetros que estén ubicados en el fichero de configuración gvHidraConfig.inc.xml . Tenemos un fichero de este tipo en cada uno de las capas que se cargarán en este orden
  - igep/gvHidraConfig.inc.xml: fichero de configuración propio del framework. No debe ser modificado ya que contiene los parámetros propios del framework.
  - igep/custom/xxx/gvHidraConfig.inc.xml: fichero de configuración de la organización. Aquí se definirán las configuraciones propias de la organización.
  - gvHidraConfig.inc.xml:fichero de configuración propio de la aplicación.
- **carga dinámica:** esta fase, se realiza una vez acabada la anterior, con lo que machacará las configuraciones que se hayan realizado. Puede ser muy útil para fijar configuraciones de forma dinámica (dependiendo del entorno de trabajo), pero es peligroso, ya que puede ser difícil depurar un error. Los ficheros por capa se ejecutarán en el siguiente orden:
  - gep/actions/gvHidraMainWindow.php: fija la configuración dinámica del framework. No debe ser modificado.
  - igep/custom/xxx/ actions/CustomMainWindow.php: fija la configuración dinámica de la organización. En este fichero se deben definir las listas y ventanas de selección de la organización.
  - actions/principal/AppMainWindow.php: fija la configuración dinámica de la aplicación. En este fichero se deben definir las listas y ventanas de selección propias de la aplicación.

2.2. Aspecto visual

2.2.1. Anatomía de una ventana gvHidra

Antes de empezar a programar vamos a entender la anatomía de una ventana gvHidra.

Una ventana de gvHidra está dividida en 4 secciones:

- **Barra superior** (1)

  En esta barra aparece alineado a la izquierda el título de la ventana, y, alineado a la derecha aparecerán, si los hay, botones tooltip. Botones tooltip son botones que efectúan acciones relacionadas con la interfaz.

- **Contenedor** (2)

  Donde se ubicará el formulario con los datos y campos con los que trabajaremos.

- **Barra inferior** (3)

  En ella, alineado a la izquierda, se ubicará el paginador y, alineados a la derecha aparecerán los botones.

- **Contenedor de pestañas** (4)

  En esta zona irán apareciendo pestañas con las que podremos cambiar el modo de trabajo (búsqueda, listado, registro...)

2.2.2. Modos de trabajo y ventanas gvHidra

La forma de trabajar con las ventanas gvHidra se ha clasificado por modos de trabajo. Definiendo modo de trabajo como la forma de representación de la información con la que vamos a trabajar. Estos modos de trabajo se verán reflejados tanto en la zona *Contenedor* (2) del panel como en la de *Contenedor de pestañas* (4), la pestaña nos indicará el modo activo.

Los modos que tenemos son tres que se corresponden con las pestañas:

1. *Búsqueda o filtro:* 
2. *Tabular o listado:* 
3. *Registro o edición:* 

Partiendo de todo lo comentado, tenemos dos formas de trabajar en gvHidra:

1. Dos modos de trabajo

   Con esto nos referimos a la forma de trabajar, en este caso partimos de una búsqueda y el resultado se muestra en una tabla o ficha, donde los datos ya son accesibles en cada uno de estos modos.

   Ej. Modo *búsqueda/tabla*

   Ej. Modo *búsqueda/ficha*

2. Tres modos de trabajo

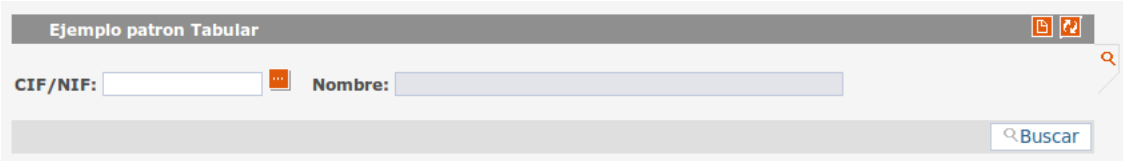
   En este caso es una combinación de los dos anteriores. Se parte de un panel de búsqueda, el resultado de dicha búsqueda se mostrará en un panel tabular. En este panel tabular los datos no son accesibles, solamente se podrá efectuar el borrado de las tuplas que se seleccionen, para inserción o modificación se pasa a un panel modo ficha.

   Ej. Modo *búsqueda/tabla/ficha*

2.2.3. Patrones de interfaz

Una vez visto la estructura de la ventana y los modos de trabajo disponibles podemos pasar a explicar los diferentes patrones de interfaz que se pueden implementar con gvHidra.

Todos los patrones tienen en común el modo búsqueda:



[NOTA: Una buena práctica es utilizar el prefijo "fil" para denominar algunos parámetros o nombres de variables que hagan referencia al panel de búsqueda.]

Patrones:

1. Tabular [19]
2. Registro [19]
3. Tabular-Registro [19]
4. Maestro-detalle [20]
5. Árbol [22]

1. **Tabular**

   Se corresponde con la forma de trabajo dos modos de trabajo, panel de búsqueda y panel tabular donde se trabajará con los datos.

   [NOTA: Una buena práctica es utilizar el prefijo "lis" para denominar algunos parámetros o nombres de variables que hagan referencia al panel tabular.]

2. **Registro**

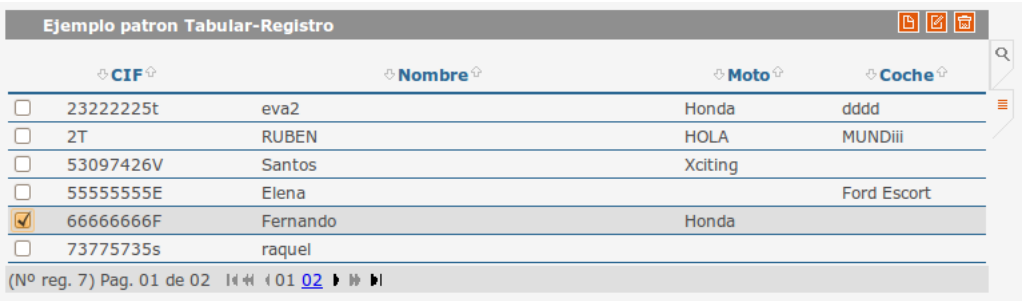
   Se corresponde con la forma de trabajo dos modos de trabajo, panel de búsqueda y panel registro donde se trabajará con los datos.

   [NOTA: Una buena práctica es utilizar el prefijo "edi" para denominar algunos parámetros o nombres de variables que hagan referencia al panel registro.]

3. **Tabular-Registro**

   Este caso corresponde a la forma de trabajo de tres modos. Un panel de búsqueda, un panel tabular donde se tendrá el resultado de la búsqueda y un panel registro donde se podrán editar los campos de las tuplas seleccionadas en el panel tabular o insertar nuevas tuplas.

   Modo tabular:



Modo registro:

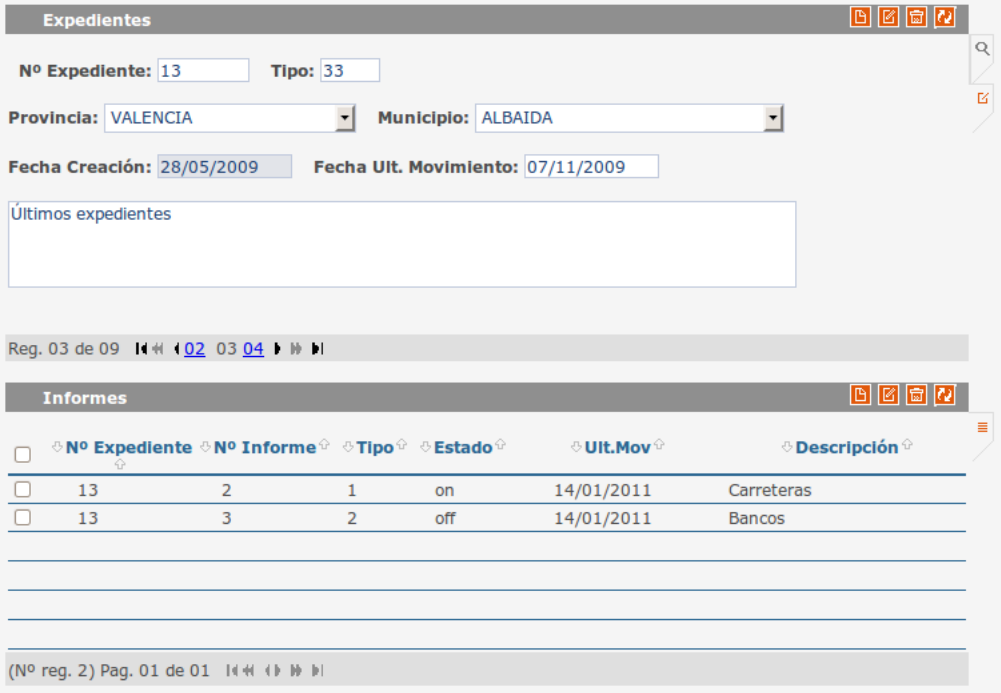
4. **Maestro-Detalle**

   La parte del maestro será del tipo dos modos de trabajo, un panel búsqueda y un tabular o registro. En cambio en la parte correspondiente al detalle se puede plantear la forma de trabajar como dos modos (tener un sólo tabular o registro) o tres modos (tabular y registro), con la excepción de que en el detalle no tenemos búsqueda.

   De esto podemos obtener diferentes combinaciones para un maestro-detalle:

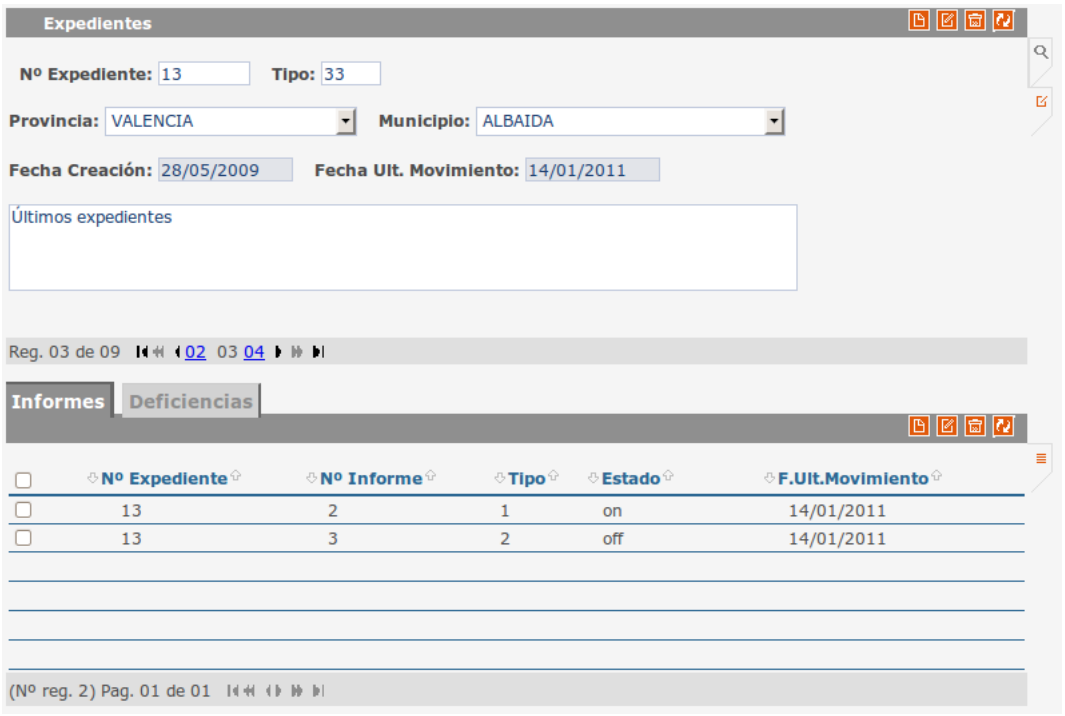
   - Maestro tabular - Detalle tabular
   - Maestro tabular - Detalle registro
   - Maestro registro - Detalle tabular
   - Maestro registro - Detalle registro
   - Maestro tabular - Detalle tabular-registro
   - Maestro registro - Detalle tabular-registro

   Ej. Maestro registro - Detalle tabular:



Además de estos patrones para el maestro-detalle contamos con un patrón más complejo, el **Maestro – N Detalles**. Este patrón es una extensión de cualquiera de los indicados anteriormente, tendremos un maestro que tendrá varios detalles asociados. Estos detalles están accesibles mediante unas solapas que aparecerán en la cabecera de la zona del detalle.

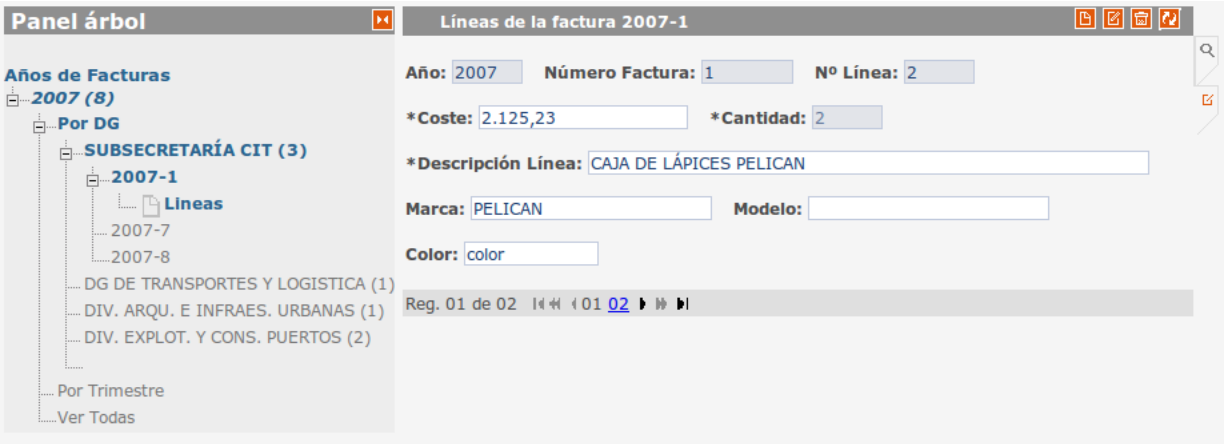
Ej. Maestro registro - N Detalles:



5. **Árbol**

   El patrón árbol se compone de dos zonas. En la zona de la izquierda encontramos el árbol en sí, una jerarquía de menús, seleccionando cualquiera de ellos accedemos a su información que aparecerá representada en la zona de la derecha. Esta información podemos decidir representarla con un patrón Tabular, Registro o Tabular-Registro.

   Ej. Árbol:



2.3. Lógica de negocio

2.3.1. Acciones y operaciones

Acciones y operaciones son dos conceptos fundamentales que hay que conocer para desarrollar con el framework.

Una acción es el proceso que se inicia tras la solicitud de un usuario (un estímulo de la IU). Las acciones están divididas en dos grupos:

1. **Acciones genéricas**

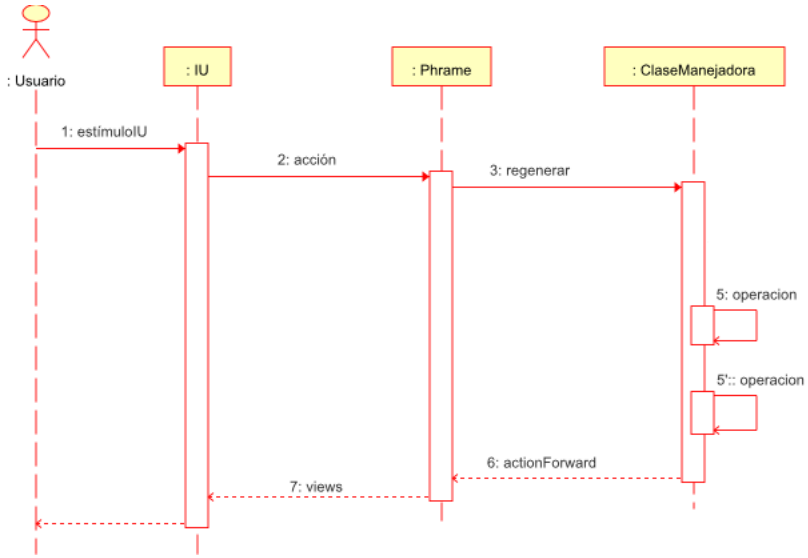
   Las acciones genéricas resuelven procesos comunes de un mantenimiento (altas, bajas, modificaciones, búsquedas...)

2. **Acciones particulares**

   Con las acciones particulares se podrán resolver necesidades concretas no resueltas por el framework.

Las **operaciones** son métodos internos del framework que serán llamados para cumplir lo que cada acción requiere, algunas de estas operaciones son públicas por lo tanto el programador podrá invocarlas desde las *acciones particulares*. El programador podrá modificar el comportamiento de por defecto de la operación sobreescribiendo con una serie de **métodos abstractos** (**virtuales**).

La siguiente imagen ilustra como se produce la comunicación entre las diferentes capas desde que un estímulo es recibido en la interfaz hasta que se resuelve. Podemos ver claramente la diferencia esencial entre acciones y operaciones.



Vamos a explicar un poco más en profundidad cada una de las acciones genéricas. Explicando tanto el objetivo global de la acción, los posibles retornos que produce, las operaciones que los componen y los métodos virtuales de cada una de estas operaciones.

Tabla 2.1. Conceptos comunes a todas las acciones genéricas

Métodos virtuales: Todas las acciones genéricas tienen una serie de métodos que el programador puede sobreescribir completando el comportamiento de la acción. Generalmente hay un método virtual “pre”, antes de la acción (inserción, modificación, borrado), y un método “post”, después de la acción.
actionForward: Acción de retorno de un método virtual. Puede tomar un valor por defecto asignado por el framework o el programador asignarle uno particular.

Tabla 2.2.

Valores de retorno de los métodos virtuales:
• **0**: Correcto, la ejecución continua. En este caso el actionForward se corresponde con la acción **gvHidraSuccess**. • **-1**: Ha habido un problema, el framework lanza un error si es problema interno de él, en cambio, si es un problema particular de la aplicación se ha de capturar el error y lanzar un mensaje propio (instrucción: *$this->showMessage('xxx');*) No se recargará la ventana. En este caso el *actionForward* se corresponde con la acción **gvHidraError**. • **actionForward**: Acción de retorno programada por el desarrollador para saltarse la ejecución de por defecto, provoca una recarga de la ventana. Común a todas las acciones genéricas salvo a la de *Recargar*, esta acción no contempla una acción de retorno particular.

2.3.1.1. Acciones genéricas

1. **Iniciar ventana**

   Esta acción se ejecutará al iniciar la ventana.

   La acción contiene la siguiente operación:

- *initWindow*: Genera todas las listas definidas para el panel y las carga en el objeto v_datosPreinsertar. Almacena el valor del módulo actual por si es la primera vez que se entra en una pantalla de dicho módulo. Su comportamiento se puede sobrecargar con el método virtual *preIniciarVentana*.

  preIniciarVentana: Método que permite realizar cualquier acción antes de que se cargue la ventana.

2. **Buscar**

   Realiza la consulta del panel basándose en los datos que hay en el panel filtro.

   La acción se compone de dos operaciones:

   - *buildQuery*: Método privado que recoge los datos del panel filtro y los añade al WHERE de la consulta del panel. Su comportamiento se puede sobrecargar con en el método virtual *preBuscar*.

     preBuscar: Método que permite realizar cualquier acción antes de que se realice la consulta.

   - *refreshSearch*: Método público que se encarga de realizar la consulta que se ha generado con el método anterior. Es importante saber, también, que este método puede ser llamado en cualquier momento para refrescar los datos de un panel, por ejemplo, después de una acción particular. Su comportamiento se puede sobrecargar con el método virtual *postBuscar*.

     postBuscar: Método que permite modificar los resultados obtenidos en la búsqueda.

     En el caso de que este método devuelva un 0 existe la posibilidad de tener un *actionForward* con la acción *gvHidraNoData* que mostrará un mensaje avisando que la consulta no ha devuelto datos, si se quiere particularizar este caso se sobrecargará la acción.

3. **Editar**

   Esta acción genérica se ejecuta cuando estamos con trabajando con tres modos. Como su propio nombre indica realiza la consulta de edición a partir de los datos seleccionados en el panel tabular.

   La acción se compone de dos operaciones:

   - *buildQueryEdit*: Método privado que recoge los datos del panel tabular con los que formará la WHERE de la consulta que nos devolverá los datos que se podrán editar. Su comportamiento se puede sobrecargar con el método virtual *preEditar*.

     preEditar: Método que permite realizar cualquier acción antes de que se realice la consulta de edición.

   - *refreshEdit*: Método público que se encarga de realizar la consulta que se ha generado con el método anterior, así obtener los datos actualizados. Es importante saber, también, que este método puede ser llamado en cualquier momento para refrescar los datos de un panel, por ejemplo, después de una acción particular. Su comportamiento se puede sobrecargar con el método virtual *postEditar*.

     postEditar: Método que permite modificar los resultados obtenidos de la consulta.

     En el caso de que no se haya seleccionado ninguna tupla para editar, aparecerá un mensaje por defecto informándonos de la situación. Para poder sobrecargar este comportamiento existe un *actionForward* con la acción *gvHidraNoData* con el que podremos hacerlo.

4. **Recargar**

   Acción genérica que actúa en un patrón maestro-detalle. Se ejecutará cuando se efectúe un cambio de maestro, recargará los detalles correspondientes.

La acción se compone de dos operaciones:

- *buildQuery*Details: Crea/recupera la instancia del panel detalle activo (puede haber N detalles activos). Llama al método *refreshDetail* de esa instancia detalle. Su comportamiento se puede sobrecargar con el método virtual *preRecargar*.

  preRecargar: Método que permite realizar cualquier acción antes de efectuar la consulta del detalle.

- *refreshDetail*: Método público que se encarga de realizar la consulta que se ha generado en el método anterior, obtener los datos del detalle a partir del maestro seleccionado. Su comportamiento se puede sobrecargar con el método virtual *postRecargar*. Es importante tener en cuenta que el programador puede llamar a este método en cualquier momento para refrescar los datos de un detalle, por ejemplo, después de una acción particular.

  postRecargar: Método que permite modificar los datos obtenidos de la consulta.

  Hay que recordar que los métodos virtuales de esta acción no tienen como retorno un actionForward programado por el desarrollador.

5. **Modificar**

   Esta acción genérica se ejecuta desde el modo de trabajo edición, cuando, tras haber editado unas tuplas, se pulsa el botón guardar.

   La acción se compone de dos operaciones:

   - *updateSelected*: Método privado que recoge los datos de pantalla y actualiza en la BD los datos modificados. Su comportamiento se puede sobrecargar con el método virtual *preModificar*.

     preModificar: Método que permite realizar cualquier acción antes de que se lleve a cabo la modificación.

   - *refreshEdit*: Método público que se encarga de actualizar los datos visualizados en el modo de trabajo edición. Su comportamiento se puede sobrecargar con el método virtual *postModificar*.

     postModificar: Método que permite utilizar los datos modificados en otras operaciones.

   - *refreshSearch o refreshDetail*: Método público que se encarga de actualizar los datos después de la operación de actualización en el panel tabular. Es importante tener en cuenta que esto implica que se lanzará el *postBuscar* o el *postRecargar*.

   La acción modificar por defecto deja el foco en el modo edición, este comportamiento se puede cambiar. El cambio se realiza en el fichero *mappings.php*, en la entrada correspondiente a modificar, en la acción *gvHidraSuccess*, cambiar en la url el panel destino (p.ej. panel=listado)

6. **Borrar**

   Esta acción genérica se dispara “típicamente” desde el modo de trabajo tabular para eliminar las tuplas seleccionadas.

   La acción se compone de dos operaciones:

   - *deleteSelected*: Recoge las tuplas seleccionadas del panel y las elimina. Esta operación se puede parametrizar haciendo uso de los métodos virtuales *preBorrar* y *postBorrar*.

     preBorrar: Método que permite realizar cualquier acción antes de que se lleve a cabo el borrado.

     postBorrar: Método que permite modificar los resultados obtenidos después del borrado.

- *refreshSearch o refreshDetail*: Método público que se encarga de actualizar los datos después de la operación de actualización en el panel tabular. Es importante tener en cuenta que esto implica que se lanzará el *postBuscar* o el *postRecargar*.

7. **Insertar**

   Esta acción genérica se dispara desde el modo de trabajo inserción cuando, tras haber introducido los datos se pulsa el botón guardar.

   La acción se compone de dos operaciones:

   - *insertData*: Recoge los datos de la pantalla y los inserta en la BD. Esta operación se puede parametrizar haciendo uso de los métodos virtuales *preInsertar* y *postInsertar*.

     preInsertar: Método que permite realizar cualquier acción antes de que se lleve a cabo la inserción.

     postInsertar: Método que permite utilizar los datos modificados en otras operaciones.

   - *refreshSearch o refreshDetail*: Método público que se encarga de actualizar los datos después de la operación de actualización en el panel tabular. Es importante tener en cuenta que esto implica que se lanzará el *postBuscar* o el *postRecargar*.

8. **Nuevo**

   Esta acción genérica se invoca para preparar la presentación de datos en pantalla antes de una inserción.

   La acción se compone de la siguiente operación:

   - *nuevo*: Realizará las operaciones de preparación de los datos y visualización previos a la inserción. Esta operación se puede sobreescribir con el método virtual *preNuevo*.

     preNuevo: Método que permite sobrecargar la acción nuevo antes de ser lanzada. Por ejemplo, cuando tenemos algún campo que tiene un valor por defecto o es calculado a partir de los valores de otros campos.

Parte II. Elementos de gvHidra

Tabla de contenidos

Capítulo 3. Elementos básicos

3.1. Estructura de aplicación

gvHIDRA, a parte de proporcionar un core (igep) en el que se encuentran todas las funcionalidades extensibles del framework, proporciona una estructura inicial de proyecto. Para empezar cualquier aplicación con la herramienta, debemos bajar la plantilla inicial de proyecto correspondiente a la versión con la que vamos a trabajar. Esta plantilla se encuentra en el paquete de la propia versión en la ubicación doc/plantilla-gvHidra.zip. Antes de descomprimir hay que advertir que los ficheros están codificados en ISO-8859-1 o ISO-8859-15, por lo que hay que tener precaución con usar algún editor que respete estas codificaciones. Otro detalle a tener en cuenta es que la plantilla contiene ejemplos que deben ser borrados.

Una vez descomprimida encontramos una estructura básica de aplicación a partir de la cual construiremos nuestra aplicación. De esta estructura destacamos:

- **gvHidraConfig.inc.xml**: donde añadir las conexiones a bases de datos, código de la aplicación y versión,...
- **actions/principal/AppMainWindow.php**: donde añadiremos las listas y ventanas de selección particulares, log, ...
- **mensajes.php**: donde añadir mensajes particulares de la aplicación.
- **include/menuModulos.xml**: define el menu principal de la aplicación. Hay que modificarlo con las acciones propias de la aplicación. También podemos usar ciertas opciones predefinidas.
- **include/menuAdministracion.xml**: definimos opciones de la 2ª columna de la pantalla principal. Siguen las mismas normas que en menuModulos.xml.
- **include/menuHerramientas.xml**: definimos opciones de la 3ª columna de la pantalla principal. Siguen las mismas normas que en menuModulos.xml.
- **include/include.php**: donde, conforme vayamos creando clases de negocio las iremos incluyendo. Los includes de ficheros "action/*" se pueden borrar.
- **include/mappings.php**: en la función ComponentesMap, excepto la llamada al constructor del padre, todo se puede borrar.
- **templates_c**: directorio de compilación de plantillas. El usuario web tiene que tener permisos de escritura.

A esta estructura tenemos que añadirle el core del framework correspondiente (directorio igep) que también se distribuye con la versión de gvHIDRA.

3.1.1. Configuración estática: gvHidraConfig.xml

La configuración básica de una aplicación gvHIDRA se concentra en los ficheros de configuración gvHidraConfig.xml. En este apartado explicaremos tanto la distribución de estos ficheros entre la arquitectura del framework como las posibilidades que ofrece. Finalmente, veremos un ejemplo de un fichero tal y como se utiliza en una aplicación.

Aunque en algunos apartados se detalla mucha información, lo realmente importante es entender el concepto general y la configuración mínima que tenemos que realizar para crear una aplicación (fichero a nivel de aplicación).

3.1.1.1. Introducción

La configuración básica de una aplicación gvHIDRA se concentra en los ficheros de configuración **gvHidraConfig.xml**. Estos ficheros de configuración (todos con el nombre mencionado) aparecerán en tres capas, una a nivel del framework

(igep), otra a nivel de custom (personalización de la organización) y finalmente, a nivel de aplicación. Por ejemplo, supongamos una aplicación realizada con gvHidra que se denomine "factur" y se encuentra dentro de la organización "tecnimap" los ficheros de configuración se ubicarán en las estructura de directorios como sigue:

- factur/igep/gvHidraConfig.inc.xml
- factur/igep/custom/tecnimap/gvHidraConfig.inc.xml
- factur/gvHidraConfig.inc.xml

La semántica de los ficheros XML vendrá determinada a través de la DTD embebida en ellos, para que a los encargados de editar y mantener esos ficheros (programadores y responsables de la aplicación correspondiente) les sea sencillo saber si los mismos están correctamente confeccionados. La finalidad de que existan distintos ficheros de configuración, es establecer una especialización jerárquica, que permita personalizar opciones de gvHidra a distintos niveles.

- Nivel de **framework** (en el ejemplo anterior, fichero factur/igep/gvHidraConfig.inc.xml): En este fichero se establecerán configuraciones generales del framework y OBLIGATORIAMENTE se establecerá el directorio de customización (customDirName).
- Nivel de **organización** (personalización o custom, en el ejemplo anterior es el fichero factur/igep/custom/tecnimap/gvHidraConfig.inc.xml): Es un fichero que incluirá elementos propios de la organización (DSNs de BDs, acciones particulares de validación, ...), además en este fichero podrá redefinirse cualquier valor establecido en el fichero a nivel de Framework.
- Nivel de **aplicación** (en el ejemplo anterior, fichero factur/gvHidraConfig.inc.xml): Utilizado para incluir configuración específica de una aplicación (DSNs, versión...), como último en el nivel jerárquico, a través de él se podrá redefinir cualquier valor establecido en el fichero a nivel de organización y de framework.

Es decir, en última instancia, las opciones que prevalecen son las que aparecen en el fichero de configuración de la aplicación. Posteriormente y a través del objeto global (el singleton ConfigFramework) podremos leer y sobrescribir dinámicamente dichas opciones.

3.1.1.2. Parámetros

A continuación vamos a detallar cada una de las opciones disponibles para la confección de los ficheros. Muchas de ellas no se deberán modificar al iniciar una aplicación, pero convien que las conozcamos:

- **applicationName**: De carácter obligatorio, establece el nombre (código) de la aplicación. La asignación normal será en la ubicación correspondiente a la aplicación.
- **appVersion**: Versión de la aplicación. La asignación normal será en la ubicación correspondiente al nivel de aplicación.
- **customTitle**: En la barra superior de color azul, a la izquierda de la fecha y hora, se ha reservado un pequeño espacio para poder incluir un texto personalizado. La asignación se realiza a través de este parámetro.
- **customDirName**: Establece el nombre del directorio de customización. En dicho directorio aparecen modificaciones o extensiones que van a ser comunes a una organización. Por ejemplo, las extensiones del Framework propias de la CIT (ventanas de selección, listas predefinidas, DSNs de consulta a BDs internas...) se ubicarán dentro de un directorio "cit.gva.es". La asignación de este valor será en el fichero a nivel de framework o de la aplicación, no pudiéndose hacer a nivel de la organización ni de forma dinámica usando el objeto global.
- **temporalDir**: Permite indicar una carpeta para la creación de algunos temporales relacionados con la seguridad (fichero de sesión, ...). La asignación de este valor será sólo en el fichero a nivel de la aplicación, no pudiéndose hacer a nivel de framework ni de la organización ni de forma dinámica usando el objeto global.

- **templatesCompilationDir**: Establece el directorio que utilizará Smarty para compilar las TPL. Si se está desarrollando algún plugin del framework, es interesante cambiar la configuración para que cada "usuario" tenga su carpeta y no haya un efecto caché mientras se desarrolla.

- **reloadMappings**: Indica si se quiere recargar el fichero de mappings en cada petición. Por defecto está habilitado, aunque habria que deshabilitarlo para entornos donde el fichero de mappings no se modifica. Este parámetro se puede definir a cualquier nivel.

- **smartyCompileCheck**: Indica si smarty tiene que comprobar si se ha modificado alguna plantilla y en caso afirmativo recargarla. Se puede definir en cualquiera de los tres xml, aunque no de forma dinámica.

- **logSettings**: Establece el nivel de detalle de los registros de auditoría.

- **queryMode**: Modo de interrogar las BDs, vease los filtros de búsqueda.

- **DSNZone**: Encerrada en esta etiqueta aparecerá toda la información relativa a las fuentes de datos. La etiqueta podrá aparecer en las tres ubicaciones, por ejemplo: a nivel de framework para establecer la configuración del DSN del log, a nivel de organización para incluir DSNs propios de la organización (listas y ventanas de selección propias, validaciones...) y finalmente, a nivel de aplicación, donde situaremos la información de conexión propia.

  - **dbDSN**: Agrupa los parámetros de conexión a un SGBD relacional. Las etiquetas están inspiradas en PEAR:MDB2.

    - *dbHost*: Host o IP donde se encuentra el servicio. En el caso de conexiónes a Oracle mediante OCI (cuando sgbd vale oracle, oci, oci8) se utilizará para establecer la entrada en tnsnames. Para los otros casos de oracle (thin y oracle-thin) debemos especificar dbHost, dbPort y dbDatabase.

    - *dbPort*: Puerto donde escucha el SGBD.

    - *dbDatabase*: Base de datos a la que nos conectamos.

    - *dbUser*: Usuario.

    - *dbPassword*: Contraseña.

  - **wsDSN**: La fuente de datos es un servicio web.

    - *uriWSDL*: URI donde puede localizarse el fichero WSDL que define el servicio web SOAP.

    - *wsUser*: Usuario

    - *wsPassword*: Contraseña

3.1.1.3. Ejemplo a nivel aplicación.

Después de esta abalancha de parámetros, para comprender mejor su utilización vamos a ver un ejemplo real de un fichero a nivel de aplicación:

```
<gvHidraConfig>
 ...
   <applicationName>factur</applicationName>
   <appVersion>1.0.0</appVersion>
   <!-- si es a nivel de organizacion, puede estar definido en el del custom-->
   <logSettings status='LOG_ALL' dnsRef='gvh_dsn_log' />

   <DSNZone>
      <dbDSN id='gvh_dsn_log' sgbd='postgres'>
         <dbHost>cualquiera.coput.gva.es</dbHost>
         <dbPort>5432</dbPort>
         <dbDatabase>saturno</dbDatabase>
         <dbUser>logUser</dbUser>
         <dbPassword>melogeo</dbPassword>
      </dbDSN>
```

```
      <dbDSN id='ora-tns' sgbd='oracle'>
         <dbHost>tns-entry</dbHost>
         <dbUser>usuario</dbUser>
         <dbPassword>pwd</dbPassword>
      </dbDSN>

      <dbDSN id='ora-thin' sgbd='oracle-thin'>
         <dbHost>bd.coput.gva.es</dbHost>
         <dbPort>1521</dbPort>
         <dbDatabase>sid</dbDatabase>
         <dbUser>usuario</dbUser>
         <dbPassword>pwd</dbPassword>
      </dbDSN>

      <wsDSN id='g_ws'>
         <wsUser>wsuser</wsUser>
         <wsPassword>wspwd</wsPassword>
      </wsDSN>
   </DSNZone>
...
</gvHidraConfig>
```

Como nota, debemos saber que podemos acceder con la clase ConfigFramework a los metodos get/set para acceder/modificar estas propiedades. Ejemplo:

```
$conf = ConfigFramework::getConfig();
$cod_apli = $conf->getApplicationName();
$conf->setCustomTitle('Tu título');
```

3.1.1.4. Recomendaciones

Hay que tener en cuenta que, dependiendo del entorno en el que estemos trabajando, puede que nos interese habilitar/deshabilitar ciertos parámetros de configuración. Típicamente, nos encontramos con el caso de entornos de producción, en estos casos interesa:

- *smartyCompileCheck*: en producción es interesante tener este valor a false para optimizar el rendimiento de nuestras aplicaciones. Con ellos aprovecharemos la cache del Smarty.
- *reloadMappings*: en producción es interesante tener este valor a false para evitar que recarge el catálogo de acciones en cada ejecución. Esto aumentará la velocidad de nuestras aplicaciones.

3.1.2. Configuración dinámica: AppMainWindow

Esta clase es propia de cada proyecto, y se crea para inicializar caracteristicas generales de la aplicación. Como se ha explicado en los puntos anteriores, supone entre otras cosas, la carga dinámica de parámetros de configuración. Este fichero no es obligatorio; aunque en la práctica podemos decir que en el 100% de los casos es necesario.

Es importante tener en cuenta que, por su definición, sólo se ejecutará la primera vez que entremos en la aplicación. A continuación describimos algunas de las funcionalidades que nos permite hacer.

3.1.2.1. Carga dinámica.

Básicamente, la carga dinámica. Si no definimos esta clase, los valores de configuración seran los definidos en los ficheros gvHidraConfig.inc.xml. Sin embargo, cuando necesitamos fijar alguno de estos parámetros en funcion de situaciones más complejas (información propia al usuario, al estado de la aplicación, al entorno de ejecución, ...), podemos usar los 'setters' de los parámetros para cambiar la configuración. Por ejemplo:

```
$conf = ConfigFramework::getConfig();

// aumentar nivel de log en desarrollo
if (ConfIgep::es_desarrollo()) {
    // Cuando estamos en desarrollo registramos todos los movimientos
    $conf->setLogStatus(LOG_ALL);
}

// mostrar cierta informacion solo para perfil informaticos
if (IgepSession::dameRol()=='R_INFORMATICOS')
    $conf->setCustomTitle('perfil-admin');
```

3.1.2.2. Acciones particulares al abrir o cerrar la aplicación

Cuando queremos hacer algo al empezar o terminar la aplicacion, la definición de esta clase nos permite acceder a estas acciones específicas. Vease el ejemplo en el punto "Código de la lógica de negocio". Para ello hay que añadir al mappings.php las siguientes líneas:

```
$this->_AddMapping('abrirAplicacion', 'AppMainWindow');
$this->_AddForward('abrirAplicacion', 'gvHidraOpenApp', 'index.php?view=igep/views/aplicacion.php');
$this->_AddForward('abrirAplicacion', 'gvHidraCloseApp', 'index.php?view=igep/views/gvHidraCloseApp.php');
```

3.1.2.3. Definir listas y ventanas de selección

Para poder añadir listas (gvHidraList) o ventanas de seleccion (gvHidraSelecctionWindow) a nuestras aplicaciones necesitamos definirlas en el constructor de esta clase. Hay más información al respecto en el apartado de las listas

3.1.2.4. Información global

Si se usan informaciones globales que no se tienen disponibles a través del patrón singleton, podemos usar los métodos guardaVariableGlobal y dameVariableGlobal de IgepSession para manejarlas. El constructor de AppMainWindow es el sitio natural donde inicializar los valores, para luego poder recuperarlos en cualquier parte de la aplicación.

3.1.3. Recomendaciones

En este apartado vamos a citar algunas de las recomendaciones que, aunque no son de obligado cumplimiento, si que nos pueden ser de gran utilidad ya que vienen recogidas de la experiencia de los desarrolladores.

3.1.3.1. Antes de arrancar

3.1.3.1.1. Configuración para desarrollo

gvHIDRA utiliza una serie de proyectos por debajo para implementar la arquitectura MVC y representar la vista. Concretamente phrame y a smarty. Estos dos proyectos tienen una serie de parámetros de configuración que interesa tener en cuenta a la hora de trabajar en una aplicación. Lógicamente no es lo mismo estar en un entorno de desarrollo que en un entorno de explotación, por ello conviene tener en cuenta cuando estamos en uno u otro para realizar la configuración adecuada.

Todo ello se resume en estas dos propiedades del fichero gvHidraConf.xml a nivel de aplicación:

- **reloadMappings**: esta propiedad que admite valores boolenos(true|false), indica si el framework debe recargar el mapeo de acciones en cada iteracción. Esto es conveniente en desarrollo, aunque ralentiza la ejecución. *En explotación esta propiedad debe estar a false.*

- **smartyCompileCheck**: esta propiedad que admite valores boolenos(true|false), indica si regeneramos la plantilla (tpl) en cada interacción. En fase de desarrollo, como cambiamos constantemente la pantalla, es conveniente que esté a true. *En explotación esta propiedad debe estar a false* para mejorar el rendimiento.

3.1.3.1.2. Codificación

El framework está en la codificación **ISO-8859-1 (Latin1)**, por lo que el IDE que utilicemos para programar, debemos configurarlo en esta codificación. Con ello evitaremos problemas con caracteres especiales. En el caso de las conexiones a BBDD, si utilizamos Oracle, debemos de definir la variable de entorno NLS_LANG=Spanish_Spain.WE8ISO8859P. En el resto no es necesario hacer nada.

3.1.3.1.3. Separar clases manejadoras por módulos

Es una buena práctica, crear carpetas dentro de los directorios actions, views y plantillas con los nombres de los módulos de la aplicación. Con esta sencilla separación física de los ficheros, será más sencillo localizar la información de un caso de uso concreto. Un ejemplo de módulos podría ser:

- Mantenimiento expedientes
- Tablas maestras
- Listados
- ...

3.1.3.1.4. Manual de Usuario Guía de Estilo

Aunque no es obligatorio, siempre conviene tener una guía para horientar a nuestros usuarios en el uso de las aplicaciones. Esta opción, abre una ventana emergente donde se explica el funcionamiento general de los distintos elementos de las aplicaciones realizadas con gvHIDRA. Se recomienda colocar en el menú de herramientas o de administración.

```
<opcion titulo="Manual Guía de Estilo" descripcion="Manual de introducción a Guía de Estilo"
 url="igep/doc/manualIGEP/indice/indice.html" imagen="menu/24.gif"
/>
```

También hay disponible una guía rápida:

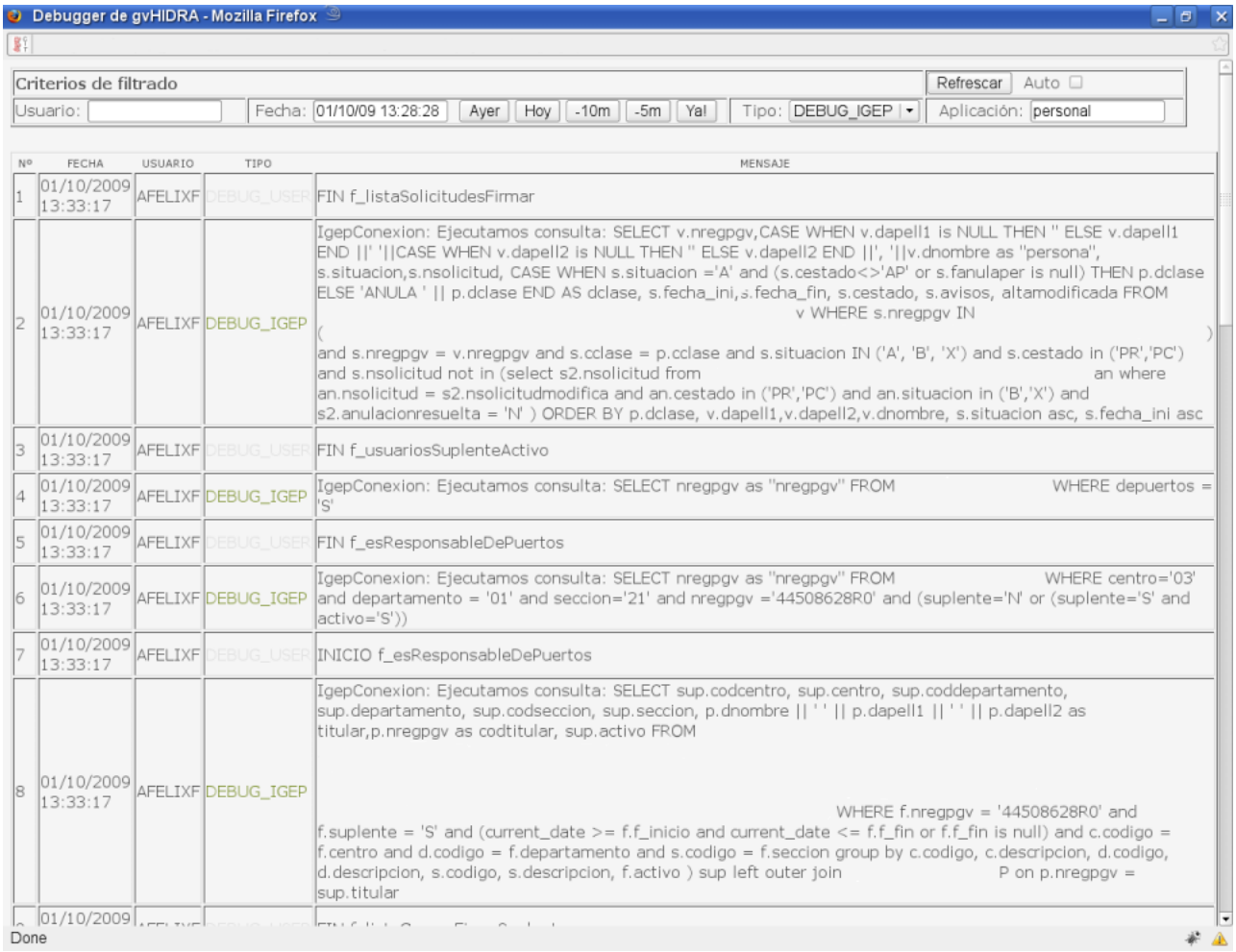
```
<opcion titulo="Guía Rápida" descripcion="Guía
      rápida uso de aplicaciones"
      url="igep/custom/cit.gva.es/doc/guiaRapida/guiaRapidaUsoAplicacionesCIT.pdf"
      abrirVentana='true' imagen="menu/24.gif" />
```

3.1.3.2. Programando...

3.1.3.2.1. Consola de Log o Debug de la aplicación

Es una de las utilidades del framework básicas. Con el siguiente código se abre una ventana emergente para consultar la información generada por la aplicación. Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Log Aplicación" descripcion="Log Aplicación"
        url="igep/_debugger.php" abrirVentana="true" imagen="menu/mensajes.gif">
    <controlAcceso>
        ...
    </controlAcceso>
</opcion>
```

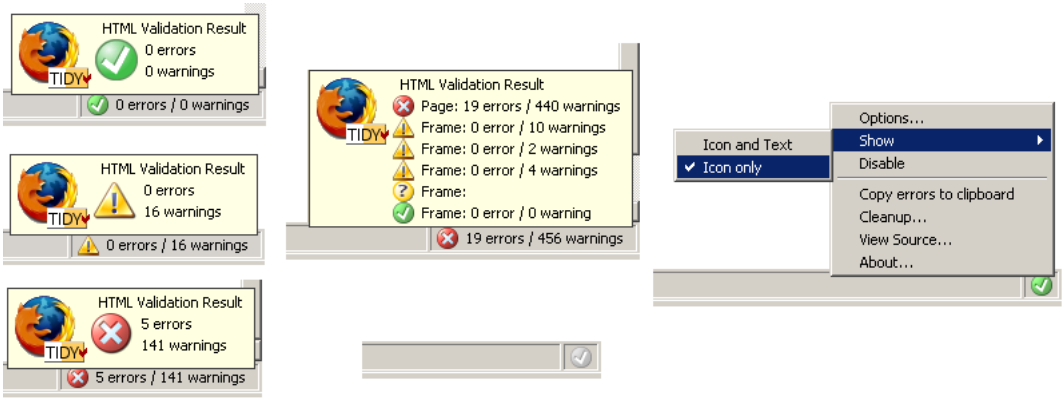


Más adelante explicaremos en profundidad el log del framework y sus posibilidades a modo de depuración y auditoría.

3.1.3.2.2. Herramientas para facilitar la depuración.

Además del propio debugger de la aplicación interesa instalarse alguna herramienta que nos permita ver el frame oculto del código HTML generado. Esto se debe a que, internamente, gvHIDRA trabaja con un frame que no es visible por el usuario, lo cual le permite, por ejemplo, realizar acciones sin recargar la página. Por todo ello, para la depuración, es fundamental instalar una herramienta que nos permita ver el contenido de dicho frame; en el caso de errores críticos, puede que sólo aparezcan en él.

Nosotros utilizamos el complemento de firefox HTML Validator, pero existen otras posibilidades. Aquí tenéis una imagen del plugin.



3.1.3.2.3. Problemas con el constructor de una clase

El framework tiene un sistema de persistencia de las clases manejadoras que permite, de forma transparente para el programador, mantener "viva" la instancia de una clase hasta que se cierre la ventana. Esto significa que, cuando se cierra una ventana y cuando se cierra la aplicación (a través de los iconos indicados), se borrarán de la sesión los datos de dicha clase. Es importante tener esto en cuenta porque si tenemos un error en el constructor, deberemos borrar la sesión antes de volver a ejecutar, ya que el constructor sólo se ejecuta la primera vez que entras a la ventana.

3.1.3.2.4. Proteger contraseñas

Se trata de una utilidad. Abre una ventana emergente donde podemos obtener el hash de una cadena (usados en servidores de web services). Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Proteger datos usando hash" descripcion="Proteger datos usando hash"
        url="igep/include/igep_utils/protectdata.php" abrirVentana="true" imagen="menu/seguridad.gif">
    <controlAcceso>
        ...
    </controlAcceso>
</opcion>
```

3.2. Breve guía para crear una pantalla

3.2.1. Introducción

A continuación vamos a explicar los pasos a seguir para crear una pantalla. Antes de llegar a este punto se tiene que tener claro el tipo de pantalla que se quiere crear.

Dependiendo del tipo de ventana seleccionado, el contenido de los ficheros será diferente. En el directorio igep/doc/plantillasTipo hay un modelo base a seguir de cada uno de los ficheros. En la siguiente tabla aparecen los diferentes patrones con los nombres de ficheros que deberemos buscar.

patrón	plantilla
Patrón Tabular / Tabular sin búsqueda	P1M2(FIL-LIS)
Patrón Tabular sin búsqueda	P1M1(LIS)
Patrón Registro	P1M2(FIL-EDI)
Patrón Registro sin búsqueda	P1M1(EDI)
Patrón Tabular-Registro	P1M2(FIL-LIS-EDI)
Patrón MD Maestro Tabular - Detalle Registro	P2M2(FIL-LIS)M1(EDI)
Patrón MD Maestro Registro - Detalle Registro	P2M2(FIL-EDI)M1(EDI)

patrón	plantilla
Patrón MD Maestro Registro - Detalle Tabular	P2M2(FIL-EDI)M1(LIS)
Patrón MD Maestro Registro - Detalle Tabular-Registro	P2M2(FIL-EDI)M1(LIS-EDI)
Patrón Árbol	

gvHidra está basado en la arquitectura MVC, por tanto al crear una pantalla tenemos que crear y modificar varios ficheros que controlen las diferentes partes de la arquitectura:

- Fichero/s del **Modelo** (directorio *actions*): se creará un fichero que controla toda la lógica de negocio del panel.
- Fichero de la **Vista** (directorios *views* y *plantillas*): se crearán unos ficheros que, por un lado controlan la interfaz antes de presentar los datos, y por otro lado, la distribución de los componentes en la pantalla.
- Fichero de **Controlador** (ficheros *mappings.php* y *menuModulos.xml*): se modificarán estos ficheros para indicar qué acción se resuelve con qué clase.

Dentro del modelo, cabe tener en cuenta que el programador interacturá con el usuario a través de dos flujos: **flujo de entrada** y **flujo de salida**. El **flujo de entrada** se compone de la información que viene de la pantalla y se gestiona con:

- *Tablas de trabajo*: Son las tablas con las que el framework va a trabajar. Es decir, las tablas sobre las que vamos a realizar operaciones con los datos que recibamos.
- *Matching*: Este método nos sirve para indicarle al framework en que se traduce cierto campo de la pantalla. Se utiliza únicamente en los flujos de entrada: construcción de la where de la operación, del Insert, Update, ...

El **flujo de salida** se compone de la información a mostrar por pantalla y se gestiona a través de las consultas que definimos en el constructor de la clase:

- *SearchQuery*: es la consulta que se lanzará al ejecutar la acción "buscar".
- *EditQuery*: es la consulta que se lanzará al ejecutar la acción "editar".

Conviene refrescar estos dos conceptos una vez concluido el siguiente ejemplo.

3.2.2. Paso a paso

Vamos a hacer un pequeño recorrido sobre como crear una pantalla, previamente se tiene que haber realizado el punto anterior de configuración de la aplicación.

1. En primer lugar tenemos que saber que tipo de pantalla necesitamos (ver tabla anterior), con ello sabremos exactamente cuantos ficheros necesitamos para la parte de *control de negocio* (directorio *actions*). Este fichero php (de aquí en adelante le llamaremos **clase manejadora**) será el que controlará la lógica de negocio, toda ella o puede compartirla apoyándose en clases de negocio adicionales cuando la aplicación lo requiera. Estas clases adicionales, por organización, las localizaremos en un directorio llamado *class* al nivel de actions.

   Los patrones los clasificamos en dos, simples y complejos. En los primeros entra el patrón tabular, patrón registro y el tabular-registro, y en los segundos, los maestro-detalle y el árbol. Esta diferencia se basa en que para los simples solamente es necesaria una clase manejadora, y para los complejos es necesaria más de una. Concretamente, en los maestro-detalle necesitaremos una clase manejadora para el maestro y una por cada detalle que tengamos, y para el árbol, se necesitará una por cada opción que vaya a tener el menú lateral.

   La clase manejadora heredará de **gvHidraForm_DB**, que contiene el comportamiento general de un mantenimiento común (inserciones, modificaciones y borrados) contra un SGBD relacional. Un ejemplo tipo de para un patrón Tabular podría ser el siguiente:

   ```
   <?php
   ```

```
/* CLASE MANEJADORA */

class TinvEstados extends gvHidraForm_DB
{

   public function __construct()
   {

      /*manejador de conexión*/
      $conf = ConfigFramework::getConfig();
      // dsnAplicacion: será el que se haya definido en gvHidraConfig.xml
      $dsn = $conf->getDSN('dsnAplicacion');

      //Las tablas sobre las que trabaja
      $nombreTablas= array('tinv_estados');
      parent::__construct($dsn,$nombreTablas);

      //La select que mostramos
      $this->setSelectForSearchQuery("SELECT cestado as \"lisCodigoEstado\",
                                          destado as \"lisDescEstado\" FROM tinv_estados");
      //El orden de presentación de los datos
      $this->setOrderByForSearchQuery("cestado");

      /* Añadimos los Matching - Correspondecias elemento TPL <-> campo de BD */
      $this->addMatching("filCodigoEstado","cestado","tinv_estados");
      $this->addMatching("filDescEstado","destado","tinv_estados");

      $this->addMatching("lisCodigoEstado","cestado","tinv_estados");
      $this->addMatching("lisDescEstado","destado","tinv_estados");
   }
}
?>
```

Del contenido de la clase podemos destacar:

- La clase hereda de **gvHidraForm_DB** (línea 5). Esto se debe a que, tal y como hemos comentado, dicha clase es la que nos proporciona el comportamiento genérico de gvHidra en relación a las acciones contra una BBDD relacional. En otros casos se puede heredar directamente de **gvHidraForm**.

- Una vez en el constructor debemos indicar la fuente de datos (líneas 12-13). Primero cargamos el fichero (**gvHidraConfig.inc.xml**) de configuración del framework, del custom y el de la propia aplicación. De esa configuración que nos ofrece el fichero cargamos el dsn que hemos definido para la aplicación, en la variable *$dsn* tendremos una referencia a la fuente de datos sobre la que se quiere trabajar. Si tenemos más de una fuente de datos, ver el capítulo Fuente de datos [110].

- Debemos llamar al constructor del padre (línea 17) e instanciar una serie de propiedades que marcarán el comportamiento del panel. Se le pasa como parámetro la conexión definida anteriormente y el nombre de las tablas que va a mantener el panel ($nombreTablas en el ejemplo).

- Debemos inicializar la consulta de la base de datos mediante el método **setSelectForSearchQuery()**. Es importante que en la select los nombres de los campos correspondan con los que luego utilizaremos en la plantilla (fichero tpl) En el ejemplo se utilizan alias para asegurar esa concordancia de nombres, esta es una medida aconsejable para evitar confusiones.

  NOTA: Ver que el alias se ha definido con un prefijo "lis" porque estamos en un patrón Tabular, la consulta nos devolverá los datos que se mostrarán en el tabular. Aconsejamos el uso de esta notación, prefijo "fil" si el campo corresponde a un panel de búsqueda, "edi" si corresponde a uno de un panel registro y "lis" si corresponde a uno de un panel tabular.

  ***IMPORTANTE**: No se recomienda usar en la select la palabra clave "DISTINCT", ya que cuando el gestor de base de datos es Oracle, gvHidra introduce una condición en el WHERE (usando la pseudocolumna ROWNUM) para limitar el número de filas, y el resultado sería incorrecto ya que primero se aplica el ROWNUM y después el DISTINCT.*

- Opcionalmente se puede modificar el WHERE y el ORDERBY de la consulta anterior mediante los métodos correspondientes. En el ejemplo vemos que se modifica el ORDERBY (línea 23), si quisiéramos modificar el WHERE que gvHidra realiza por defecto utilizaríamos el siguiente método **setWhereForSearchQuery()**.

- El siguiente paso a realizar es dar la correspondencia entre los nombres de los campos (en la consulta) y los nombres en la plantilla (tpl). Cuando los datos son devueltos desde el panel a la capa de negocio, se necesita realizar una conversión, ya que los nombres no tienen porque coincidir (independizamos la presentación de la fuente de datos). Para realizar este proceso (que hemos denominado matching) se debe utilizar un método heredado llamado

addMatching. En la llamada a este método le hemos de indicar 3 parámetros, que por este orden, corresponden a: nombre del campo en la TPL, nombre del campo en la consulta (el nombre del campo o el alias utilizado en la SELECT) y nombre de la tabla de la BD a la que pertenece. Para evitar problemas con los nombres de columnas entre distintos SGBD (oracle y postgresql por ejemplo), conviene usar siempre el 'as' en los nombres de columna, usando un alias preferiblemente en minúsculas.

ATENCIÓN: Todo lo expuesto se corresponde con un panel con sólo *dos modos* (tabular o registro). Si se tiene un panel con *tres modos* (tabular-registro) se deben inicializar además las siguientes propiedades:

- Utilizar el método **addCamposClave()** para indicar que campos forman parte de la clave primaria del mantenimiento. Estos campos son los que se utilizarán para realizar el paso del modo de trabajo TABULAR al modo FICHA (registro).

- Inicializar la SELECT que contiene la consulta a realizar en el panel de FICHA con el método **setSetSelectForEditQuery()**. Es decir, la select que se lanzará cuando tras seleccionar una o más tuplas en tabular y pulsamos el botón editar, nos pasará a la FICHA.

- Opcionalmente se puede rellenar la WHERE y el ORDERBY de la edición con los métodos correspondientes, **setWhereForEditQuery()** y **setOrderByForEditQuery()**.

Hasta aquí tenemos una clase manejadora muy sencillita para una pantalla muy básica. Podemos tener la necesidad de que la interfaz sea un poco más dinámica, por ejemplo, tener listas y campos dependientes, explicado al final de este capítulo y en el capítulo 4 Elementos de pantalla avanzados [90].

También es importante que el programador sepa de la existencia de algunos métodos que puede sobrecargar para dotar de cierto comportamiento particular a las funciones genéricas de gvHidra. Por ejemplo, puede ser muy útil realizar validaciones previas a una operación en la base de datos o especificar restricciones complejas en la WHERE de una consulta dependiendo de los valores seleccionados por el usuario. Estos métodos son los explicados en el capítulo 1, punto 2 Lógica de negocio [22].

Antes de pasar al siguiente paso vamos a dar un ejemplo de estos métodos. Concretamente, vamos a forzar que cuando se inserten tuplas en nuestro ejemplo, la descripción aparezca siempre en mayúsculas.

```
public function preInsertar($objDatos)
{
   while($v_datos=$objDatos->fetchTupla())
   {
      $v_datos['descEstado'] = strtoupper($v_datos['descEstado']);
      $objDatos->setTupla($v_datos);
   }
   return 0;
}//Fin de PreInsertar
```

2. El siguiente paso es indicar al controlador que clase es la encargada de realizar las distintas operaciones del panel y donde se debe ir dependiendo del resultado de la operación realizada. Para ello editamos el fichero **mappings.php** (directorio include de la aplicación) y añadimos las operaciones que correspondan a dicho panel.

   Para cada operación necesitamos definir quien gestiona cada operación, y donde debe redirigirse en cada posible respuesta de esa operación, si es correcta o no la respuesta.

   Con la función **_AddMapping** indicamos que clase se encarga de gestionar cada operación. De esta función los parámetros que nos interesan son los dos primeros. Con el primero indicamos el nombre de la operación, este nombre debe seguir el siguiente patrón *nombreClaseManejadora__nombreAccion*, los dos nombres están separados por dos subguiones. El segundo parámetro indica el nombre de la clase manejadora que tratará las operaciones (en el ejemplo TinvEstados).

   En *nombreAccion* nos podemos encontrar las siguientes palabras reservadas:

   - **iniciarVentana**: Acción que se lanzará cuando se quiera incluir en un panel algo que tenga que ser cargado desde BD o alguna comprobación previa a la carga de la ventana. Si se produce algún error (gvHidraError) en este momento nos redirigirá a la pantalla de entrada de la aplicación (igep/views/aplicacion.php).

Por ejemplo poner en un panel de búsqueda una lista desplegable cargada desde BD.

- **buscar**: Acción que normalmente se dispara desde los paneles de filtro. Comprueba si la búsqueda tiene parámetros y lanza la SELECT definida en la clase manejadora para la búsqueda.

- **operarBD**: Acción que realiza las tres operaciones básicas de un mantenimiento: Insertar, Borrar y Modificar. Esta acción es exclusiva para trabajar con patrones de un solo modo, patrón Tabular o patrón Registro con o sin búsqueda.

- **nuevo**: Acción que se lanza al pulsar al botón de nuevo registro en un panel. Se utiliza para cargar listas u otros componentes, inicializar datos antes de que el usuario empiece la inserción de datos.

- **editar**: Acción que se lanza al pulsar el botón modificar de un panel, concretamente cuando nos encontramos en un Tabular-Registro. Se lanzará la consulta definida para la edición en la clase manejadora correspondiente.

- **insertar**: Acción exclusiva para cuando se trabaja con un patrón Tabular-Registro. Se lanza cuando se va a insertar el registro nuevo.

- **modificar**: Acción exclusiva para cuando se trabaja con un patrón Tabular-Registro. Se lanza cuando se van a guardar las modificaciones hechas en el panel.

- **borrar**: Acción exclusiva para cuando se trabaja con un patrón Tabular-Registro. Se lanza cuando se va a efectuar la operación de borrado.

- **cancelarTodo**: Acción que, como su propio nombre indica, cancela todo, es decir, elimina el contenido de la última consulta y de la última edición.

- **cancelarEdicion**: En este caso, esta acción solamente elimina el contenido de la última edición.

- **recargar**: Acción exclusiva para ventanas maestro-detalle. Acción que se lanza al paginar en un maestro para recargar los detalles.

Ahora tenemos que definir las redirecciones dependientes de las respuestas a la operación, para ello tenemos el método **_AddForward**. Este método necesita de tres parámetros, siendo el primero el mismo que para la función _AddMapping, el segundo parámetro será una palabra clave que identifica la respuesta a la operación, y el tercer parámetro será la ruta donde se redirigirá la respuesta, normalmente será a un fichero de la presentación del directorio *views*.

Las *acciones genéricas*, que corresponden con las acciones enumeradas antes, tienen unos retornos fijos que dependerán del resultado de la operación, las palabras clave de respuesta para estas acciones son **gvHidraSuccess**, **gvHidraError**, **gvHidraNoData**, significando: todo correcto, ha habido error, no hay datos, respectivamente. Las *acciones particulares* tendrán tantos retornos como el programador crea conveniente. A continuación mostramos la parte del mappings que correspondería a nuestro ejemplo:

```
<?php

class ComponentesMap extends gvHidraMaps
{
        /**
         *      constructor function
         *      @return void
         */

        function ComponentesMap ()
        {
            parent::gvHidraMaps();
            ...
            $this->_AddMapping('TinvEstados__iniciarVentana', 'TinvEstados', '', 'IgepForm', 0);
            $this->_AddForward('TinvEstados__iniciarVentana', 'gvHidraSuccess', 'index.php?view=views/tablasMaestras/p_estados.php');
            //
            $this->_AddForward('TinvEstados__iniciarVentana', 'gvHidraError', 'index.php?view=igep/views/aplicacion.php');

            $this->_AddMapping('TinvEstados__buscar', 'TinvEstados', '', 'IgepForm', 0);
            $this->_AddForward('TinvEstados__buscar', 'gvHidraSuccess', 'index.php?view=views/tablasMaestras/p_estados.php&panel=listar');
            $this->_AddForward('TinvEstados__buscar', 'gvHidraError', 'index.php?view=views/tablasMaestras/p_estados.php&panel=buscar');
```

```
            $this->_AddForward('TinvEstados__buscar', 'gvHidraNoData', 'index.php?view=views/tablasMaestras/p_estados.php&panel=buscar');

            $this->_AddMapping('TinvEstados__operarBD', 'TinvEstados', '', 'IgepForm', 0);
            $this->_AddForward('TinvEstados__operarBD', 'gvHidraSuccess', 'index.php?view=views/tablasMaestras/p_estados.php&panel=listar');
            $this->_AddForward('TinvEstados__operarBD', 'gvHidraError', 'index.php?view=views/tablasMaestras/p_estados.php&panel=listar');
            $this->_AddForward('TinvEstados__operarBD', 'gvHidraNoData', 'index.php?view=views/tablasMaestras/p_estados.php&panel=buscar');
            ...
        }
}
```

3. A continuación debemos crear un archivo php en el directorio **views** que controle la presentación. En él se definen las asignaciones de los datos a la plantilla (tpl). Si se trata de un comportamiento genérico, hay que instanciar la clase **IgepPantalla** (línea 3) que define el comportamiento general de una pantalla. En la línea 5 se carga la clase **IgepPanel**, que nos permite tener accesibles las funciones de acceso al panel, pasándole dos parámetros, el primero será el nombre de la *clase manejadora* que corresponde a ese panel, y el segundo, *smty_datosTabla*, es el nombre de una variable smarty que se habrá definido en la tpl correspondiente al panel, en el siguiente punto se explica. Una vez aquí tenemos que activar las pestañas laterales que nos permitirán cambiar de modo (en el ejemplo búsqueda/tabular), esto lo haremos con el método **activarModo**, al que se le pasan dos parámetros, el primero es el panel al que hará referencia ("fil" -> panel búsqueda, "lis"-> panel tabular, "edi"-> panel registro), y el segundo parámetro es nombre de un parámetro que indica el estado de la pestaña ("estado_fil"->pestaña filtro, "estado_lis"->pestaña tabular, "estado_edi"->pestaña registro), si está activa o no.

   Una vez definido todo tenemos que agregar el panel a la ventana, esto lo haremos con el método **agregarPanel()**. Y por último, ya solo nos queda mostrar la tpl en pantalla, para ello utilizamos el método **display()**, que es un método propio de smarty, siendo la variable $s una instancia de la clase Smarty_Phrame.

   A continuación mostramos un fichero de ejemplo de una ventana con un panel de dos modos:

```
<?php
//Creamos una pantalla
$comportamientoVentana = new IgepPantalla();

//Creamos un panel
$panel = new IgepPanel('TinvEstados',"smty_datosTabla");

//Activamos las pestañas de los modos que tendremos en el panel
$panel->activarModo("fil","estado_fil");
$panel->activarModo("lis","estado_lis");

//Agregamos el panel a la ventana
$comportamientoVentana->agregarPanel($panel);

//Realizamos el display
$s->display('tablasMaestras/p_estados.tpl');
?>
```

4. Ahora vamos a diseñar la plantilla (tpl) de la pantalla en el directorio plantillas de la aplicación. En la tpl diseñaremos como quedará al final la pantalla, con una combinación de plugins propios de gvHidra, etiquetas propias de smarty y etiquetas html haremos el diseño. Las plantillas tendrán todas una estructura común a partir de la cual podremos ir particularizando nuestra pantalla.

   En ella creamos los componentes necesarios para nuestra pantalla, ajustando todos los parámetros de cada plugin con los nombres dados en la clase manejadora y las acciones correspondientes del mappings. Lo veremos más detallado en el punto 3 Diseño de pantalla con smarty/plugins [52].

   Precauciones a tener en cuenta a la hora de diseñar una tpl:

   - Los identificadores de campos pueden estar en cualquier combinación de mayúsculas y minúsculas, aunque no debe haber dos donde sólo cambie esto (es decir, case-insensitive).
   - Hay que evitar poner atributos o tags innecesarios, ya que eso se traduce en una página HTML de mayor peso. Por ejemplo, si una columna es oculta, no tiene sentido indicar su tamaño, obligatoriedad, ...
   - Los carácteres especiales como acentos o eñes conviene ponerlos usando entidades html [http://www.etsit.upm.es/%7Ealvaro/manual/manual.html#6], para que no hay problemas con la codificación.
   - No conviene usar el carácter '_' en las tpls. Mejor darle otro nombre mediante alias en la select.

- Cuando en la tpl exista una ficha ({CWFicha}) habrá que distribuir los campos dentro de ella. En principio si son pocos campos con un simple salto de línea (<br />) se podrían dibujar, pero cuando se complica más se creará una tabla con los parámetros que se indican en el ejemplo, y luego distribuir los campos en celdas.

- Delante del primer campo de la celda no debemos poner ningún espacio en blanco ( )

```
<tr>
    <td>
        {CWCampoTexto ...
    </td>
</tr>
```

Solamente se pondrá en el caso de que se sitúen dos campos en una misma celda; así dejaríamos un espacio entre ellos. Ejemplo:

```
<tr>
    <td>
      {CWCampoTexto ...} {CWCampoTexto ...}
    </td>
</tr>
```

Ejemplo completo de una plantilla, hay que destacar que es importante **no repetir identificadores de elementos de pantalla** (nombres de los campos) dentro de una misma TPL, como vemos en el ejemplo, los conceptos estado y descripción estado corresponden a los campos filCodigoEstado y filDescEstado

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
 {CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=$smty_customTitle}
  {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
 {/CWBarra}

 {CWMarcoPanel conPestanyas="true"}

 <!--*********** MODO fil ******************-->
   {CWPanel id="fil" action="buscar" method="post" estado="$estado_fil" claseManejadora="TinvEstados"}
     {CWBarraSupPanel titulo="Estados de los bienes"}
       {CWBotonTooltip imagen="04" titulo="Limpiar campos" funcion="limpiar" actuaSobre="ficha"}
     {/CWBarraSupPanel}
     {CWContenedor}
       {CWFicha}
         <br />
         {CWCampoTexto nombre="filCodEstado" size="3" editable="true" textoAsociado="Código" dataType=$dataType_TinvEstados.filCodEstado}
         <br /><br />
         {CWCampoTexto nombre="filDescEstado" size="10" editable="true" textoAsociado="Descripción" dataType=$dataType_TinvEstados.filCodigoEstado}
         <br /><br />
       {/CWFicha}
     {/CWContenedor}
     {CWBarraInfPanel}
       {CWBoton imagen="50" texto="Buscar" class="boton" accion="buscar" }
     {/CWBarraInfPanel}
   {/CWPanel}


 <!-- ***************** MODO lis ***********************-->
   {CWPanel id="lis" tipoComprobacion="envio" action="operarBD" method="post" estado="$estado_lis" claseManejadora="TinvEstados"}
     {CWBarraSupPanel titulo="Estados de los bienes"}
       {CWBotonTooltip imagen="01" titulo="Insertar registros" funcion="insertar" actuaSobre="tabla"}
       {CWBotonTooltip imagen="02" titulo="Modificar registros" funcion="modificar" actuaSobre="tabla"}
       {CWBotonTooltip imagen="03" titulo="Eliminar registros" funcion="eliminar" actuaSobre="tabla"}
     {/CWBarraSupPanel}
     {CWContenedor}
       {CWTabla conCheck="true" conCheckTodos="true" id="Tabla1" datos=$smty_datosTabla}
         {CWFila tipoListado="false"}
           {CWCampoTexto nombre="lisCodEstado" textoAsociado="Cód. Estado." editable="nuevo" size="2" dataType=$dataType_TinvEstados.lisCodEstado}
           {CWCampoTexto nombre="lisDescEstado" textoAsociado="Desc. Estado." editable="true" size="12" dataType=$dataType_TinvEstados.lisDescEstado}
         {/CWFila}
         {CWPaginador enlacesVisibles="3"}
       {/CWTabla}
     {/CWContenedor}
     {CWBarraInfPanel}
       {CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
       {CWBoton imagen="42" texto="Cancelar" class="boton" accion="cancelar" action="cancelarTodo"}
     {/CWBarraInfPanel}
   {/CWPanel}

 <!-- ***************** PESTAÑAS ***********************-->
   {CWContenedorPestanyas}
     {CWPestanya tipo="fil" estado=$estado_fil}
     {CWPestanya tipo="lis" estado=$estado_lis}
   {/CWContenedorPestanyas}
 {/CWMarcoPanel}
{/CWVentana}
```

5. Registramos el fichero de actions con la nueva clase en el fichero **include.php** de la aplicación que se encuentra en el directorio *include*. Podemos hacerlo con un include o usando la carga dinámica de clases explicada en el capítulo 4 punto Carga dinámica de clases [106].

6. Finalmente incluimos la ventana correspondiente en *include/menuModulos.xml* que contiene las entradas de menú.

```
<opcion titulo="Estados" descripcion="Mantenimiento de estados de bienes" url="phrame.php?action=TinvEstados__iniciarVentana"/>
```

Una vez cubiertos estos pasos se puede probar la ventana entrando a la aplicación. Suerte!

3.3. Menú de una aplicación

3.3.1. Funcionamiento del menú

La pantalla de inicio de la aplicación está formada por tres partes, y cada una de ellas se define, de izquierda a derecha, en los ficheros menuModulos.xml, menuHerramientas.xml y menuAdministracion.xml que están en la carpeta include de la aplicación. Las tres siguen la misma sintaxis y tienen las mismas opciones, que veremos a continuación. A continuación tenemos un ejemplo:

El plugin CWPantallaEntrada, lee los tres ficheros para crear la pantalla de inicio. El objetivo de cada fichero es:

- **menuModulos.xml**: Define la jerarquía de módulos y opciones de la aplicación, que se utilizará tanto en la pantalla principal como en el menú desplegable de las ventanas.
- **menuHerramientas.xml**: Lista de herramientas disponibles para esta aplicación.
- **menuAdministracion.xml**: Define las herramientas para la administración de la aplicación.

La aplicacion puede dividirse en módulos, etiqueta <**modulo**>, aunque normalmente sólo lo harán aquellas de tamaño grande; una aplicacion de tamaño mediano o pequeño constará de un sólo módulo. Dentro de cada módulo pueden aparecer tanto ramas, etiqueta <**rama**>, como opciones, etiqueta <**opcion**>; las ramas expresan submenús y las opciones son las hojas o elementos que se asocian con una acción o pantalla.

La etiqueta <**controlAcceso**> sirve para controlar que partes de la aplicación son visibles o no, en función del rol o de los módulos asignados al usuario (esa información se extrae de la clase IgepSession a través de la sesion). Si este nodo no aparece, cualquier usuario validado podrá acceder. Este nodo debe ser siempre el primer hijo del nodo al que quiera asociarse (NÓTESE que en el caso de opcion, debe reconvertirse la etiqueta XML y ser necesario una de apertura y otra de cierre).

A continuación se relacionan el conjunto de etiquetas utilizadas son:

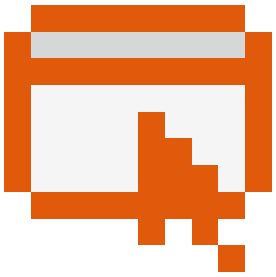
La imágenes que pueden aparecer asociadas a una opción del menú se encuentran ubicadas en igep/images/menu

- **<modulo>...</modulo>**

  Agrupación de opciones y ramas. Sólo se pueden poner a primer nivel.

  Atributos:

  - **titulo**: Título de la opción, será la cadena presentada en el menu y en la pantalla principal.
  - **descripcion**: Se corresponde con la cadena que se muestra en el menú cuando detenemos el cursor encima.

- **imagen**: (Opcional) Imagen asociada en el menú. Será una ruta relativa a un fichero gráfico en la carpeta igep/images. Si instanciamos el parámetro, deberemos comprobar que la imagen existe. Si no especificamos la imagen para el módulo, por defecto se utilizará la siguiente imagen 

- **<rama>...</rama>**

  Opción dentro de un módulo que contiene opciones o más ramas

  Atributos:

  - **titulo**: Título de la opción, será la cadena presentada en el menu y en la pantalla principal.
  - **descripcion**: Se corresponde con la cadena que se muestra en el menú cuando detenemos el cursor encima.
  - **imagen**: (Opcional) Imagen asociada en el menú. Será una ruta relativa a un fichero gráfico en la carpeta igep/images. Si instanciamos el parámetro, deberemos comprobar que la imagen existe. Si no especificamos este parámetro por defecto se utilizará la siguiente imagen 

- **<opcion />**

  Opción que ya enlazará con la ventana correspondiente. NÓTESE que en el caso de que se quiera control de acceso debe reconvertirse la etiqueta XML y ser necesario una de apertura y otra de cierre (<opcion><controlAcceso>...</controlAcceso></opcion>)

  Atributos:

  - **titulo**: Título de la opción, será la cadena presentada en el menú y en la pantalla principal.
  - **descripcion**: Se corresponde con la cadena que se muestra en el menú cuando detenemos el cursor encima.

- **imagen**: (Opcional) Imagen asociada en el menú. Será una ruta relativa a un fichero gráfico en la carpeta igep/images. Si instanciamos el parámetro, deberemos comprobar que la imagen existe. Si no especificamos la imagen para la opción por defecto se utilizará la siguiente imagen 

- **url**: Indicaremos la clase PHP encargada de manejar la opción (se añade internamente la cadena view).

- **ventana**: Parámetro que utilizaremos cuando queramos que la url se nos abra en una ventana diferente de la aplicación, pondremos ventana="true". Si no aparece se abrirá en la misma ventana de la aplicación.

- **<controlAcceso>...</controlAcceso>**

  Bloque para definir los roles y módulos de usuarios. Aparecerá inmediatamente después del módulo, rama u opción que haya que controlar.

- **<rolPermitido>**

  Define el rol para acceder al nodo. Se hace uso de tantas etiquetas <rolPermitido> como roles puedan acceder a ese nodo.

  Atributos:

  - **valor**: Identificador del rol

- **<moduloPermitido>**

  Módulo de acceso. No tiene nada que ver con la etiqueta módulo que se ha definido antes, y que sirve para agrupar las opciones en la aplicación.

  <*moduloPermitido* /> Pueden imitar el comportamiento de los roles o ser más restrictivos. Si sólo se especifica el módulo, los usuarios que lo tengan asignado podrán acceder.

  <*moduloPermitido*>...</*moduloPermitido*> Si además se especifica una lista de valores, se comprobará si el usuario tiene asignado un módulo, y si lo tiene, el valor del mismo debe aparecer entre los especificados en la etiqueta <valorModulo> .

  Atributos:

  - **id**: Identificador del módulo

- <**valorModulo**>

  Aparecerá una etiqueta por cada valor al que se le dará acceso a la opción.

  Atributos:

  - **valor**: valor del módulo

3.3.1.1. Ejemplo con el fichero completo

```
<?xml version="1.0" encoding="iso-8859-1"?>

<menu aplicacion="nombreAplicacion">

    <modulo id="1" titulo="modulo Uno" imagen="menu.gif">
        <controlAcceso>
            <rolPermitido valor="INFORMATICO"/>
            <rolPermitido valor="SUPERNENA"/>
            <rolPermitido valor="MIEMBROIGEP"/>
            <moduloPermitido id="P_TRAMITA" />
            <moduloPermitido id="david">
                <valorModulo valor=" " />
            </moduloPermitido>
            <moduloPermitido id="DIASEMANA">
                <valorModulo valor="LUNES" />
                <valorModulo valor="MARTES" />
            </moduloPermitido>
        </controlAcceso>
        <opcion imagen="menu.gif" titulo="tOpcion1.1" descripcion="dOpcion1.1" url="URLOpcion1-1.php" />
        <opcion imagen="menu.gif" titulo="tOpcion1.2" descripcion="dOpcion1.1" url="URLOpcion1-1.php" />
        <rama titulo="ramaL2" id="r2">
            <opcion imagen="menu.gif" titulo="tOpcion2.1" descripcion="dOpcion2.1" url="URLOpcion2-1.php">
            </opcion>
            <opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2" url="URLOpcion2-1.php"/>
            <rama titulo="ramaL3" id="r3">
                <opcion imagen="menu.gif" titulo="tOpcion3.1" descripcion="dOpcion3.1" url="URLOpcion3-1.php" />
                </opcion>
            </rama>
        </rama>
        <opcion imagen="menu.gif" titulo="tOpcion1.4" descripcion="dOpcion1.4" url="http://www.google.es" />
    </modulo>

    <modulo id="2" titulo="modulo Segundo" imagen="menu.gif">
        <opcion imagen="menu.gif" titulo="tOpcion1.1" descripcion="dOpcion1.1" url="URLOpcion1-1.php" />
        <rama titulo="ramaL2" id="r2">
            <opcion imagen="menu.gif" titulo="tOpcion2.1" descripcion="dOpcion2.1" url="URLOpcion2-1.php">
                <controlAcceso>
                   <rolPermitido id="INFORMATICO"/>
                   <moduloPermitido id="P_TRAMITA"/>
                 </controlAcceso>
            </opcion>
            <opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2" url="URLOpcion2-1.php" />
            <rama titulo="ramaL3" id="r3">
                <controlAcceso>
                    <moduloPermitido id="COLORES">
                        <valorModulo valor="ROJO" />
                        <valorModulo valor="AZUL" />
                    </moduloPermitido>
                </controlAcceso>
                <opcion imagen="menu.gif" titulo="google" descripcion="Buscador" url="http://www.google.es"/>
            </rama>
        </rama>
    </modulo>

</menu>
```

3.3.1.2. Ejemplos con control de acceso

Ejemplo: Opción sin control de acceso.

```
<opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2" url="URLOpcion2-1.php" />
```

Ejemplo: Opción con control de acceso.

```
 <opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2" url="URLOpcion2-1.php">
    <controlAcceso>
        <moduloPermitido id="COLORES">
                <valorModulo valor="ROJO" />
                <valorModulo valor="AZUL" />
        </moduloPermitido>
     </controlAcceso>
 </opcion>
```

En dicho nodo se incluye tanto el tratamiento de los roles de usuario como el de los módulos.

- **Rol**: se hace uso de tantas etiquetas <rolPermitido> como roles puedan acceder a ese nodo, cada rol se identifica por el atributo valor.

  Ejemplo:

```
<opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2" url="URLOpcion2-1.php">
   <controlAcceso>
        <rolPermitido valor="DESARROLLO_GVHIDRA" />
   </controlAcceso>
</opcion>
```

- **Módulo de acceso**: Pueden imitar el comportamiento de los roles o ser más restrictivos. Si sólo se especifica el módulo, los usuarios que lo tengan asignado podrán acceder. Si además se especifica una lista de valores, se comprobará si el usuario tiene asignado un módulo, y si lo tiene, el valor del mismo debe aparecer entre los especificados.

  Ejemplo: Opción con lista de valores, el usuario debe tener asignado el módulo COLORES y deberá tener uno de los dos valores, ROJO o AZUL para que se permita el acceso.

```
<opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2" url="URLOpcion2-1.php">
    <controlAcceso>
        <moduloPermitido id="COLORES">
            <valorModulo valor="ROJO" />
            <valorModulo valor="AZUL" />
        </moduloPermitido>
    </controlAcceso>
</opcion>
```

3.3.2. Opciones predefinidas para el menu

Opciones que pueden usarse en los ficheros xml de configuración de menús, y que están implementadas en gvHidra.

3.3.2.1. Manual de Usuario Guía de Estilo

Abre una ventana emergente donde se explica el funcionamiento general de los distintos elementos de las aplicaciones realizadas con gvHIDRA. Se recomienda colocar en el menú de herramientas o de administración.

```
    <opcion titulo="Manual Guía de Estilo" descripcion="Manual de introducción a Guía de Estilo"
            url="igep/doc/manualIGEP/indice/indice.html" imagen="menu/24.gif"
    />
```

También hay disponible una guía rápida:

```
<opcion titulo="Guía Rápida" descripcion="Guía rápida uso de aplicaciones"
         url="igep/custom/cit.gva.es/doc/guiaRapida/guiaRapidaUsoAplicacionesCIT.pdf"
         abrirVentana='true' imagen="menu/24.gif"
 />
```

3.3.2.2. Consola de Log o Debug de la aplicación

Abre una ventana emergente para consultar la información generada por la aplicación. Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Log Aplicación" descripcion="Log Aplicación"
        url="igep/_debugger.php" abrirVentana="true" imagen="menu/mensajes.gif">
   <controlAcceso>
      ...
   </controlAcceso>
</opcion>
```

3.3.2.3. Proteger contraseñas

Abre una ventana emergente donde podemos obtener el hash de una cadena (usados en servidores de web services). Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Proteger datos usando hash" descripcion="Proteger datos usando hash"
        url="igep/include/igep_utils/protectdata.php" abrirVentana="true" imagen="menu/seguridad.gif">
   <controlAcceso>
      ...
   </controlAcceso>
</opcion>
```

3.3.3. Autenticación y autorización (módulos y roles)

3.3.3.1. Introducción

Los módulos y roles es la forma usada para controlar el acceso a las distintas partes de un aplicación. No hay que confundir estos módulos con la etiqueta <*modulo*> usados a nivel de los menús para agrupar funcionalidades.

3.3.3.1.1. Módulos

Los módulos representan permisos que un usuario tiene en una aplicación determinada. Los módulos se asignan cuando se lleva a cabo una asociación entre un usuario y una aplicación, y se cargan en el inicio de la aplicación. Cada módulo tiene un código, una descripción y opcionalmente puede tener un valor. Cada usuario puede tener asignado uno o varios módulos, y para cada uno puede modificar su valor. Algunos usos habituales de módulos son el controlar si un usuario puede acceder a una opción de menú, o si puede ver/editar un campo determinado de una pantalla, o para guardar alguna información necesaria para la aplicación (departamento del usuario, año de la contabilidad, ...).

También suelen usarse para definir variables globales para todos los usuarios, aunque en este caso es recomendable asignarlos a roles (ver apartado siguiente).

Ejemplo:

- Todos los usuarios que consulten la aplicación PRESUPUESTARIO deben tener el módulo M_CONSUL_PRESUPUESTARIO, es decir, nadie que no tenga ese módulo asociado podrá acceder al apartado de listados de la aplicación
- Sólo el personal de la oficina presupuestaria tendrá acceso a la manipulación de datos, es decir el módulo M_MANT_PRESUPUESTARIO
- Sólo técnicos y el jefe de la oficina presupuestaria tendrán tendrán acceso a la entrada de menú "control de crédito". Pues serán aquellos usuarios que tengan el módulo M_MANT_PRESUPUESTARIO, con el valor TECNICO o el valor JEFE.
- Usuarios:
  - Sandra (Administrativa de otro departamento que consulta la aplicación): Tiene el módulo M_CONSUL_PRESUPUESTARIO
  - Juan (Administrativo): Tiene el módulo M_MANT_PRESUPUESTARIO, (bien sin valor, o con el valor PERFIL_ADMD)
  - Pepe (Técnico): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO
  - Pilar (Jefa de la oficina presupuestaria): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_JEFE
- Módulos:
  - Nombre: M_CONSUL_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las consultas de la aplicación de presupuestario
    - Valores posibles: <COMPLETAR>
  - Nombre: M_MANT_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las opciones de mantenimiento de la aplicación de presupuestario
    - Valores posibles: PERFIL_ADMD, PERFIL_TECNICO o PERFIL_JEFE

3.3.3.1.2. Roles

Los roles representan el papel que el usuario desempeña en una aplicación y a diferencia de los módulos, cada usuario sólo puede tener uno y no tienen valor. ¿Entonces para que queremos los roles si podemos hacer lo mismo usando módulos sin valor? Podemos ver los módulos como los permisos básicos, y los roles como agrupaciones de módulos.

Realmente los roles se utilizan para facilitar la gestión de usuarios. Si sólo usamos módulos y por ejemplo tuviéramos 30 módulos, seria muy difícil clasificar a los usuarios ya que seguramente cada uno tendría una combinación distinta de módulos, y cada vez que hubiera que dar de alta un usuario tendríamos que tener un criterio claro para saber cuales de los módulos asignar.

Con roles es más simple, ya que es sencillo ver los permisos de cualquier usuario (viendo el role suele ser suficiente), y para añadir nuevos usuarios normalmente solo necesitaremos saber su role. Para que esto sea fácil de gestionar, tendríamos que tener algún mecanismo que nos permitiera asignar módulos a roles, y que los usuarios con un rol "heredaran" estos módulos. Lo más flexible seria tener esta información en tablas aunque también se podría empezar haciéndolo directamente en PHP (o ver abajo módulos dinámicos):

```
if ($role == 'admin') {
   // combinacion 1
   $modulos[] = array('borrarTodo'=>array('descrip'=>'borra todo',));
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion11'=>array('descrip'=>'opcion 11',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   $modulos[] = array('opcion13'=>array('descrip'=>'opcion 13',));
   ...
} elseif ($role == 'gestor') {
   // combinacion 2
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   ...
} elseif ($role == '...') {
...
```

De esta forma, añadir un nuevo usuario de tipo administrador consistiría simplemente en indicar su role, en vez de tener que asignarle los N módulos que tienen los administradores.

La solución más flexible seria usar sólo módulos para controlar cualquier característica que pueda ser configurable por usuario, y asignar los módulos a los roles de forma centralizada (bien en base de datos o en el inicio de la aplicación). Asi el programador solo tendría que tratar con los módulos, y el analista con los módulos de cada role.

El ejemplo anterior usando roles podría ser:

- módulos: M_CONSUL_PRESUPUESTARIO, M_MANT_PRESUPUESTARIO (ambos sin valor)
- roles:
  - PERFIL_ADMD (módulo M_MANT_PRESUPUESTARIO), asignado a Juan
  - PERFIL_TECNICO (módulo M_MANT_PRESUPUESTARIO), asignado a Pepe
  - PERFIL_JEFE (módulo M_MANT_PRESUPUESTARIO), asignado a Pilar
  - PERFIL_OTROS (módulo M_CONSUL_PRESUPUESTARIO), asignado a Sandra

En este caso las dos soluciones tienen una complejidad similar, aunque las diferencias serán más evidentes conforme aumenten el número de módulos y usuarios. Si por ejemplo decidiéramos que ahora todos los técnicos han de tener un nuevo módulo X, sin usar roles tendríamos que añadir ese módulo a todos los usuarios que tuvieran el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO; usando roles seria simplemente añadir el módulo X al role PERFIL_TECNICO.

3.3.3.2. Uso en el framework

El primer ejemplo de uso de módulos puede encontrarse en la creación de los ficheros xml que da lugar a la pantalla de inicio de la aplicación, en dichos ficheros se utiliza la etiqueta "controlAcceso" para indicar que módulos necesita tener asignados un usuario para acceder a la opción. Por ejemplo:

```
<opcion titulo="Prueba del árbol" descripcion="Probando el árbol"
        url="phrame.php?action=IgepPruebaArbol__iniciarVentana" imagen="menu/24.gif">
   <controlAcceso>
        <moduloPermitido id="M_INTRANET"/>
   </controlAcceso>
```

```
</opcion>
```

Con el párrafo anterior de XML, se indica que la entrada de la aplicación "Prueba del árbol" sólo estará disponible para aquellos usuarios que cuenten entre sus módulos el M_INTRANET. También se puede hacer el control usando un role.

Los módulos podemos usarlos en otros sitios, y podemos acceder a ellos a través de los métodos:

Tabla 3.1. Listado de Métodos 1

método	descripción
IgepSession::hayModulo(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo
IgepSession::dameModulo(<modulo>)	Información del módulo
ComunSession::dameModulos()	Todos los módulos (estáticos) asignados al usuario (se recomienda usar el método con el mismo nombre de IgepSession)

Cuando queremos usar módulos que no estén permanentemente asignados a un usuario, sino que la asignación dependa de otras circunstancias que puedan ir cambiando durante la ejecución de la aplicación, la asignación no la haremos en el inicio de la aplicación. Para poder añadir o eliminar módulos en tiempo de ejecución, y conseguir, entre otras cosas el poder hacer accesibles u ocultar opciones de menú dinámicamente, podemos además usar:

Tabla 3.2. Listado de Métodos 2

método	descripción
IgepSession::anyadeModuloValor(<modulo>, <valor>=null, <descripcion>=null)	Añade un nuevo módulo al usuario
IgepSession::quitaModuloValor(<modulo>, <valor>=null)	Quita el módulo al usuario; si se le pasa valor, sólo lo quita si el valor del módulo coincide con el valor pasado
IgepSession::hayModuloDinamico(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo dinámico
IgepSession::dameModuloDinamico(<modulo>)	Información del módulo dinámico
IgepSession::dameModulosDinamicos()	Todos los módulos dinámicos asignados al usuario
IgepSession::dameModulos()	Todos los módulos asignados al usuario

Estos módulos les llamaremos módulos dinámicos. A efectos del framework, no hay diferencia entre los módulos cargados inicialmente y los módulos dinámicos. El plugin **CWPantallaEntrada** ya incluye dicha funcionalidad, por lo que si en alguno de los ficheros XML aparece una opción como esta:

```
<opcion titulo="Prueba de módulo dinamico"
      descripcion="Ejemplo de activación y descativación de opciones en ejecución"
      url="http://www.gvpontis.gva.es" imagen="menu/24.gif">
   <controlAcceso>
      <moduloPermitido id="MD_GVA"/>
   </controlAcceso>
</opcion>
```

hasta que en algún punto de nuestro código PHP no realicemos una llamada como esta:

```
IgepSession::anyadeModuloValor("MD_GVA")
```

la opción permanecerá oculta.

Si después de habilitarla queremos volver a ocultarla, basta con ejecutar la opción:

```
IgepSession::quitaModuloValor("MD_GVA")
```

Podemos obtener el role del usuario con el método **IgepSession::dameRol()**.

Nota

Restricciones en los menús

Para que los cambios sean efectivos de forma visual en los menús, es necesario que se reinterprete el código XML, cosa que sólo se hace cuando se pasa por la pantalla principal de la aplicación, mientras tanto NO será actualizada la visibilidad/invisibilidad de las distintas ramas, módulos y/o opciones. Por tanto, convendrá también añadir en las acciones que correspondan al menú, comprobación explícita de que se tienen los módulos/roles necesarios para el acceso.

3.4. Diseño de pantalla con smarty/plugins

3.4.1. ¿Qué es un *template*?

Es un fichero de texto con extensión TPL. La idea original de Smarty es que fuese una plantilla o TEMPLATE (de ahí su extensión). En el caso de gvHidra, se intentaba que la Web se pareciese lo máximo posible a un modelo Cliente-Servidor, con una estructutra de capas para distanciar lo máximo posible el diseño.

El fichero .tpl, es una plantilla en la que se diseña la parte de presentación combinando algunas etiquetas HTML (opcionalmente) con etiquetas propias de Smarty y elementos que denominamos "plugins" creados para gvHidra. Smarty actúa como motor de esas plantillas, y renderiza las etiquetas y plugins en código HTML y Javascript.

Smarty está implementado en PHP, por lo que se emplea ese lenguaje para procesar la presentación, es decir, los plugins lo utilizan para la lógica de presentación de una plantilla. Esto nos permite obtener una clara separación entre la aplicación lógica y el contenido en la presentación. No hay problema entre la lógica y su plantilla bajo la condición que esta lógica sea estrictamente para presentación.

3.4.2. Cómo realizar un *template* (tpl)

Los elementos que aparecen entre llaves dentro de la tpl pueden ser tanto plugins propios de gvHidra como etiquetas propias de Smarty. Principalmente utilizaremos plugins para diseñar la presentación de la aplicación, aunque en momentos puntuales puede ser que necesitemos el uso de alguna etiqueta de smarty.

Tenemos dos tipos de plugins:

1. *Plugin block:* Son plugins que pueden anidar otros plugins. Se componen de una etiqueta de apertura y una de cierre. En la etiqueta de apertura es donde se parametrizará el plugin.

   Ej. **{CWTabla** conCheck="true" seleccionUnica="true" id="Tabla1" datos=$smty_datosTabla**}**

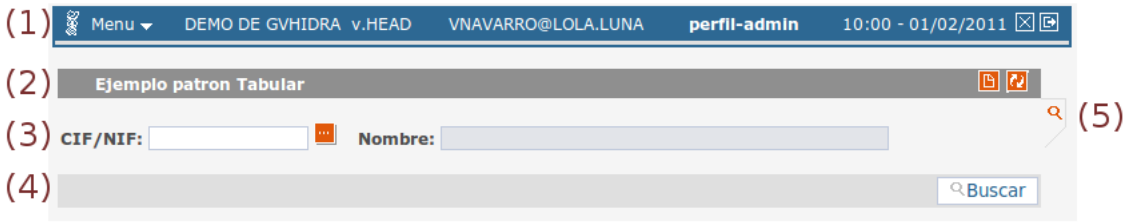
   ...

   {/CWTabla}

2. *Plugin function:* Son plugins independientes, pueden estar dentro de un plugin block. Estos solamente tienen una etiqueta, y será en ella donde parametrizaremos el plugin.

   Ej. **{CWCampoTexto** nombre="lisCoche" editable="true" size="10" textoAsociado="Coche"**}**

La tpl deberá tener una estructura válida, es decir, hay ciertos plugins que no se deben colocar de forma aleatoria, sino que tienen su orden. En el punto 1.1 del capítulo 2 Anatomia de una ventana gvHidra se explicó las zonas que existen en una pantalla gvHidra, podemos decir que en la tpl se refleja esa distribución de la pantalla.



La imagen anterior nos muestra una pantalla, ella estará englobada por el plugin **{CWVentana}** ... **{/CWVentana}**

- **{CWBarra} ... {/CWBarra}** *(1)*

  En este bloque tendremos el menú de la aplicación, si lo hay, con el plugin **{CWMenuLayer}**. A continuación diversa información, opcional, que podrá ser dada con los parámetros del plugin CWBarra. Los botones del final de la barra son fijos de gvHidra, la X vuelve al menú principal y el otro nos sacará de la aplicación.

  Dentro de este bloque podremos colocar botones tooltip, si son necesarios, que se corresponden con el plugin **{CWBotonTooltip}**

- **{CWMarcoPanel} ... {/CWMarcoPanel}** *(2) (3) (4) (5)*

  Este plugin engloba todo lo que consideramos panel de la pantalla. Dentro de él se diferenciarán los distintos modos de trabajo, cada modo de trabajo, *(2) (3) (4)*. irá englobado por las etiquetas **{CWPanel} ... {/CWPanel}**.

- **{CWBarraSupPanel} ... {/CWBarraSupPanel}** *(2)*

  Dentro de este bloque podremos insertar el plugin **{CWBotonTooltip}** para crear los botones de la derecha.

- **{CWContenedor} ... {/CWContenedor}** *(3)*

  Aquí es donde diseñaremos la parte principal de la pantalla, donde distribuiremos los campos, listas... donde ya utilizaremos unos plugins u otros dependiendo de si queremos diseñar un tabular o un registro.

  Tabular Necesitaremos insertar el plugin {CWTabla} al que se le pasarán ciertos parámetros, entre ellos está el más importante, *datos*, que será el que contendrá todos los datos.

  Tendrá una estructura como la siguiente:

  {CWTabla} **{CWFila}**

  ...

  {/CWFila}

  {/CWTabla}

  Registro Necesitaremos insertar el plugin {CWFichaEdicion} al que se le pasarán ciertos parámetros, entre ellos está el más importante, *datos*, que será el que contendrá todos los datos.

  Tendrá una estructura como la siguiente:

  {CWFichaEdicion} **{CWFicha}**

  ...

  {/CWFicha}

{/CWFichaEdicion}

Un caso especial será cuando estamos diseñando la búsqueda, en este caso tenemos un {CWFicha} pero sin necesidad de estar dentro de un bloque {CWFichaEdicion}.

Búsqueda Tendrá una estructura como la siguiente:

{CWFicha} ...

{/CWFicha}

En esta zona es donde conoceremos la totalidad de los datos con los que se va a trabajar, por eso tenemos la excepción de que el plugin **{CWPaginador}** se ubica dentro de este bloque aunque físicamente luego se vea en el bloque del **{CWBarraInfPanel}**.

- **{CWBarraInfPanel} ... {/CWBarraInfPanel}** *(4)*

  En esta barra se incluirán, si se quieren, los botones que se corresponden con el plugin **{CWBoton}**.

- **{CWContenedorPestanyas} ... {/CWContenedorPestanyas}** *(5)*

  Este bloque engloba las pestañas que aparecerán en la pantalla, tendremos que insertar un plugin **{CWPestanya}** por cada pestaña que queramos.

Hay una serie de plugins que serán obligatorios y otros opcionales, que añadiremos o quitaremos según la funcionalidad que se le quiera dar a la pantalla. En la documentación de los plugins [168] se encuentra la definición de cada uno, así como la serie de argumentos que contendrán.

[*NOTA: En la carpeta 'igep/doc/plantillasTipo' hay un ejemplo de plantilla para cada tipo de pantalla a crear.*]

[*NOTA: Tener en cuenta a la hora de diseñar la plantilla la resolución de pantalla de los usuarios. Si por ejemplo tienen 800x600 nos caben unas 12 líneas en una tabla, a diferencia de las 32 que nos caben en una resolución 1280x1024* .]

Vamos a presentar la estructura obligatoria para los principales modos de trabajo:

3.4.2.1. Estructura para mantenimientos generales

```
{CWVentana ...}
   {CWBarra ...}
      si existe menú habrá {CWMenuLayer}
   {/CWBarra}

   {CWMarcoPanel ...}

      PANEL DE FILTRO
      {CWPanel id="fil" ...}
         {CWBarraSupPanel ...}
            Si existen botones ToolTip {CWBotonToolTip ...}
         {/CWBarraSupPanel}
         {CWContenedor}
            {CWFicha}

            {/CWFicha}
         {/CWContenedor}
         {CWBarraInfPanel}
            Si existen botones {CWBoton ...}
         {/CWBarraInfPanel}
      {/CWPanel}

      PANEL DE EDICIÓN
      {CWPanel id="edi|ediMaestro|ediDetalle" ... }
         {CWBarraSupPanel ...}
            Si existen botones ToolTip {CWBotonToolTip ...}
         {/CWBarraSupPanel}
      {CWContenedor}
         {CWFichaEdicion ...}
            {CWFicha}

            {/CWFicha}
         {/CWFichaEdicion}
      {/CWContenedor}
```

```
      {CWBarraInfPanel}
         Si existen botones {CWBoton ...}
      {/CWBarraInfPanel}
   {/CWPanel}

   PANEL LISTADO
   {CWPanel id="lis|lisMaestro|lisDetalle" ...}
      {CWBarraSupPanel ...}
         Si existen botones ToolTip {CWBotonToolTip ...}
      {/CWBarraSupPanel}
      {CWContenedor}
         {CWTabla ...}
            {CWFila ...}

            {/CWFila}
         {/CWTabla}
      {/CWContenedor}
      {CWBarraInfPanel}
         Si existen botones {CWBoton ...}
      {/CWBarraInfPanel}
   {/CWPanel}

   {CWContenedorPestanyas ...}
      aparecera un {CWPestanya ... } por cada pestaña que tengamos.
   {/CWContenedorPestanyas}
 {/CWMarcoPanel}
{/CWVentana}
```

3.4.2.2. Mejorando el aspecto visual

Vamos a dar unas indicaciones a la hora de la distribución y aspecto de los campos en el panel para intentar mejorar su visualización.

1. **Distribución de campos dentro de un CWFicha.**

   Para la distribución de los campos en la ficha nos tendremos que apoyar en el lenguaje HTML. Etiquetas como
 (saltos de línea),   (espacios en blanco) y <table> (tablas) nos ayudarán a distribuir los campos en el panel. Si nos decantamos por la distribución mediante tablas, utilizar el estilo asociado llamado "*formularios*", ver punto Personalizando el estilo [71].

2. **Tabulación entre los campos.**

   Para facilitar la navegación entre los campos de una ventana, todos los componentes disponen de un parámetro **tabindex** que permite indicar el orden en el que se quieren recorrer. Por convenio, recomendamos asignar valores de 10 en 10 entre los componentes, esto facilita la incorporación de nuevos componentes sin problemas.

```
{CWCampoTexto tabindex="10" ...}
{CWLista tabindex="20" ...}
```

   Los campos no editables no entrarán en esta navegación con el tabulador. Si, por una acción de interfaz un campo se convierte en editable, el sistema le asignará valor del parámetro "tabindex" de la tpl. Otra opción es que el programador desde una acción de interfaz cambie dinámicamente el tabindex de un componente. Para ello dispone del método **setTabIndex**.

```
$objIU->setTabIndex('field1',30);
```

3. **Resaltado de filas en un CWTabla.**

   Para poder resaltar una fila en una tabla de gvHidra tenemos que indicárselo en la construcción de la misma (típicamente en una acción buscar método postBuscar). Para ello utilizaremos el método del objDatos **setRowColor**, el cual tiene dos parámetros: la fila y el color. Actualmente, los posibles valores de color son: 'alerta', 'aviso', 'error' y 'sugerencia'.

   Ejemplo:

```
function postBuscar($objDatos)
{
   //Obtenemos la matriz de datos
   $m_datos = $objDatos->getAllTuplas();

   foreach($m_datos as $indice =>$tupla)
   {
      //Si el tipo de la tupla es urgente lo marcamos con un color.
      if($tupla['tipo']=='URGENTE')
         $objDatos->setRowColor($m_datos[$indice],'alerta');
```

```
            else
                $objDatos->setRowColor($m_datos[$indice],'none');
        }

        //Guardamos la matriz de datos modificada
        $objDatos->setAllTuplas($m_datos);
    }
```

3.4.3. Documentación de los plugins

La documentación respecto a los plugins, descripción, parámetros y uso la podemos ver en el Apéndice B Documentación de plugins gvHIDRA [168].

3.5. Código de la lógica de negocio

En este punto entraremos en detalle del código de la lógica de negocio explicada en el punto Lógica de negocio del capítulo 2 [22].

3.5.1. Operaciones y métodos virtuales

Vamos a adentranos en el código para entender mejor las *operaciones* de gvHidra y sus *métodos virtuales*, explicadas en el punto 2 del capítulo 1 Lógica de negocio.

Las acciones buscar, insertar, editar, borrar, modificar... son acciones generales que no contemplan las reglas de negocio particulares de una aplicación. Por ejemplo, si tenemos un panel que muestra facturas, el programador puede utilizar la acción borrar para cuando un usuario lo desee pueda borrar una factura. Pero, ¿y si quiere que sólo se borren las facturas que no estén abonadas? Para ello debemos modificar el comportamiento general de la acción. Esta modificación se hará en las clases de negocio (*actions*) sobreescribiendo uno de los *métodos abstractos* (*virtuales*) vistos anteriormente con el código particular que se desee introducir.

En general, tenemos un método para antes de realizar la operación, que nos permite cancelarla (*pre-validación*) y un método para después de la operación (*post-validación*), que se puede utilizar para realizar operaciones en otras tablas.

Métodos virtuales:

1. preIniciarVentana [58]
2. preBuscar [64], postBuscar [64]
3. preInsertar [59], postInsertar [59]
4. preNuevo [60]
5. preModificar [60], postModificar [60]
6. preBorrar [61], postBorrar [61]
7. preEditar [62], postEditar [62]
8. preRecargar [63], postRecargar [63]
9. openApp [64], closeApp [64]

Todos estos métodos tienen como parámetro una instancia de la clase IgepComunicaUsuario que proporcionará al programador un acceso total a los datos que intervienen en la operación. Este acceso a los datos vendrá dado por los siguientes métodos disponibles en cualquiera de los métodos virtuales citados:

- getValue / setValue
- getOldValue
- nextTupla
- currentTupla
- fetchTupla / setTupla
- getAllTuplas / setAllTuplas
- getAllOldTuplas
- getOperacion / setOperacion

Para trabajar campo por campo, en la variable $objDatos ...:

```
//Devuelve el valor del campo de la fila actual

$objDatos->getValue('nombreCampo');

//Devuelve el valor del campo antes de ser modificado
//(ojo: devuelve el valor que tiene en la BD)

$objDatos->getOldValue('nombreCampo');

//Fija el valor del campo de la fila actual

$objDatos->setValue('nombreCampo','nuevoValor');

//Devuelve el registro activo sobre el origen de datos actual (cursor)

$tupla = $objDatos->currentTupla();

//Mueve el cursor a la fila siguiente

$objDatos->nextTupla();
```

Para trabajar tupla a tupla:

```
//Devuelve un vector con el contenido de la tupla actual
//y mueve el cursor interno

$tupla = $objDatos->fetchTupla();

//Fija el contenido de la fila activa con el valor deseado.

$objDatos->setTupla($tupla);
```

Para trabajar directamente con la matriz de datos:

```
//Devuelve una matriz que contiene todas las tuplas que intervienen
//en la operación

$m_datos = $objDatos->getAllTuplas();

//Fija la matriz con las tuplas que intervienen en la operación

$objDatos->setAllTuplas($m_datos);

//Devuelve la matriz de registros original correspondiente al origen de datos
//pasado como argumento (datos inserción, modificación, borrado) o el
//preestablecido.

$m_datos = $objDatos->getAllOldTuplas();
```

Para trabajar con una matriz de datos concreta (ver acceso a datos [67]):

```
//Fija la operación que será origen de los datos con los que se trabajará
//El parámetro será la operación de la cual se quiere la matriz
//  - 'visibles': Matriz de datos visibles en la pantalla.
//  - 'insertar': Matriz de datos que serán insertados en la BD
//  - 'actualizar': Matriz de datos que van a ser modificados en la BD.
//  - 'borrar': Matriz de datos que serán borrados de la BD.
//  - 'seleccionar': Matriz de datos de la/s tupla/s seleccionada/s.
//  - 'buscar': Matriz de datos que se utilizarán para la búsqueda.
//  - 'postConsultar': Matriz de datos después de una búsqueda.
//  - 'seleccionarPadre': En patrones maestro-detalle, matriz de datos que
//        nos dará la tupla seleccionada del padre.
//  - 'iniciarVentana': Matriz que contiene todos los datos del REQUEST
//  - 'external': Matriz de datos con campos no relacionados con la matriz
```

```
//       de datos.

$objDatos->setOperacion('insertar');
```

Este método, **setOperacion**, se debe utilizar en el caso de que nos encontremos en una acción particular, ya que con él fijaremos la operación sobre qué conjunto de datos queremos trabajar.

Puede darse el caso que para realizar ciertas validaciones necesitemos datos de otro panel. Para esto tenemos una clase que permite el acceso a la información de dicho panel, que se encuentra almacenada en la sesión (IgepSession).

```
// Te devuelve un campo de la tupla seleccionada.
$campo = IgepSession::dameCampoTuplaSeleccionada('clasePanel','nomCampo');

// Te devuelve la tupla seleccionada.
$tuplaSeleccionada = IgepSession::dameTuplaSeleccionada('clasePanel');

// Te devuelve el valor de una propiedad particular de la clase
$propiedad = IgepSession::dameVariable('clasePanel','propiedad');
```

Otro de los métodos que podemos sobreescribir es el encargado de cargar parámetros en las búsquedas. Concretamente es el método setSearchParameters, con él se puede indicar a la capa de negocio algunas condiciones adicionales a la where que el framework no añade por defecto.

Por otra parte, desde el método abstracto (virtual) se pueden ejecutar sentencias SQL que actuarán sobre la conexión que tenga el panel por defecto, para ello tenemos el método **consultar** (*$this->consultar($select);*) que se ejecutará con los parámetros definidos en IgepConexion.

Vamos a describir algunos ejemplos para los diferentes *métodos virtuales*.

- **Iniciar Ventana**

  Método que se ejecuta al iniciar la ventana. Por ejemplo, si tenemos una misma clase manejadora que se accede desde dos entradas de menú diferentes, agregando un parámetro en la llamada podremos saber en qué opción de menú ha entrado.

  Ejemplo Tenemos dos entradas de menú que llaman a la misma clase, una con el parámetro tipoAcceso con valor A y otro con valor B.

```
<opcion titulo="Ventana Sin Modificar" descripcion="No permitimos modificar"
url="phrame.php?action=Clase__iniciarVentana&amp;tipoAcceso=A"/>

<opcion titulo="Ventana Con Modificar" descripcion="Permitimos modificar"
url="phrame.php?action=Clase__iniciarVentana&amp;tipoAcceso=B"/>
```

Teniendo esto en cuenta, podemos comprobar qué opción de menú ha seleccionado para acceder y, por ejemplo, activar o no ciertos controles.

```
public function preIniciarVentana($objDatos) {
    $tipoAcceso = $objDatos->getValue('acceso');
    if ($tipoAcceso=='A') {
        //No puede modificar...
    }
    else{
        //Puede modificar...
    }
    return 0;
}
```

Veamos otro ejemplo donde controlamos que se visualicen unos campos o no en el panel de búsqueda, dependiendo de si el usuario es administrador o no. En primer lugar creamos un método **preIniciarVentana** en la claseManejadora:

```
public function preIniciarVentana($objDatos) {
        $admin = IgepSession::dameRol();
        //Almacenamos en una variable de instancia si es administrador o no.
        if($admin=='Administrador')
                $this->esAdministrador = 1;
        else
                $this->esAdministrador = 0;
        return 0;
}
```

Ahora en el views tenemos que asignar la variable a la tpl:

```
$admin = IgepSession::dameVariable('<claseManejadora>','esAdministrador');
$s->assign('smty_administrador',$admin);
```

Ahora en la tpl utilizaremos la lógica de smarty para poder ocultar los campos:

```
{if $smty_administrador eq 1}
        {CWCampoTexto nombre="cif" editable="true" size="5" textoAsociado="CIF"}
        {CWCampoTexto nombre="nombre" editable="true" size="40" textoAsociado="Nombre"}
{/if}
```

Con esto tendremos que el campo cif y nombre solo aparecerán si el usuario es administrador.

- **Insertar**

  Por defecto se insertan todas las tuplas que el usuario ha introducido en el panel. Con los métodos **preInsertar**, poder validar si se continua o no con la operación, y **postInsertar**, poder realizar operaciones posteriores a la inserción, controlaremos el funcionamiento particular de este proceso.

  Ejemplo En *preInsertar* se comprueba que todas las facturas que se han introducido correspondan a un proveedor válido.

```
public function preInsertar($objDatos)
{
   while($tupla = $objDatos->fetchTupla())
   {
        //Comprobamos que el cif y el orden pertenecen o a un proveedor o a una factura
        $str_selectCAj = "SELECT count(1) as \"cuenta\" FROM tcom_proveed
        WHERE nif = '" .$tupla['cif']."' AND orden = ".$tupla['orden'];
        $resCAj = $this->consultar($str_selectCAj);
        if($resCAj[0]['cuenta']!=1)
        {
           //Error de validación
           $this->showMensaje('APL-24');
           return -1;
        }
   }
   return 0;
}
```

Otro ejemplo típico es el del cálculo de secuencias. En el siguiente ejemplo mostramos como se calcula el número de secuencia de nuevas facturas dependiendo del año.

```
public function preInsertar($objDatos)
{
    //Para que cuando insertas dar el autonumérico
    $secFacturas['anyo']=$objDatos->getValue('anyo');
    $numEntrada = $this->calcularSecuencia('tinv_facturas','nfactura',$secFacturas);
    do
    {
        $objDatos->setValue('nfactura',$numEntrada);
        $numEntrada++;
    } while($objDatos->nextTupla());
    return 0;
}//Fin de PreInsertar
```

El *postInsertar* se ha utilizado para mandar un correo después de la inserción del registro en la tabla:

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $indice = key($m_datos);
  if ($objDatos->getValue('dgral')!='' && $objDatos->getValue('cserv')!=''))
        $nombreServDgral = $conexionOracle->consultar("select ddg as \"nombreDgral\",dservc as \"nombreServicio\"
                                                       from vcmn_organoso where cdgo='".$objDatos->getValue('dgral')."'
                                                       and cservo='".$objDatos->getValue('cserv')."'");

  //Obtenemos los datos de la clave informática,el número reset y el nombre del expediente
  if ($objDatos->getValue('codexp')!='')
  {
        $claveinf = $conexionOracle->consultar("select numero_reset as \"numreset\",cianyo as \"cianyo\",cidg as \"cidg\",
                                               cinum as \"cinum\",citipo as \"citipo\",cinumtipo as \"cinumtipo\",
                                               objeto as \"nombre_exp\"
                                               from vopu_todos_exped where claveseg='".$objDatos->getValue('codexp')."'");
        if(count($claveinf)<=0)
        {
                $this->showMensaje('APL-18');
                return -1;
        }
  }

  //Para poder obtener los datos del usuario de la sesión
  $datosusuario=IgepSession::dameDatosUsuario();

  //Formamos el asunto del mensaje:
  $asunto=$objDatos->getValue('aplicacion')."/".$datosusuario['nombre'];
```

```
              ...

  return parent::envioCorreo($m_datos,$rol,$indice,$datosusuario,$asunto,$texto);

}//Fin de postInsertar
```

Otro ejemplo de postInsertar puede ser que al insertar en una tabla maestro queramos que se realice una inserción en una tabla detalle dependiente (entidad débil). Tras realizar esto, por ejemplo, podemos decidir salir de la pantalla con un actionForward de salida, indicado para este caso especial.

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $error = $conexionOracle->operar("INSERT INTO tabla2 VALUES (1,2)");
  if($error==-1)
  {
    $this->obj_errorNegocio->setError("APL-15","TinvLineas2.php","postInsertar");
    return -1;
  }
  else
    return $objDatos->getForward('salir');
}//Fin de postInsertar
```

- **Nuevo**

  Tenemos el método **preNuevo**, que cuando accedemos a un panel en el que vamos a introducir datos nuevos para insertar, nos va a permitir preparar algunos datos o visualización de datos, previos a la inserción, por ejemplo, cuando tenemos algún campo que tiene un valor por defecto o es calculado a partir de los valores de otros campos.

  Ejemplo Tenemos un campo secuencial que no debe ser introducido por el usuario.

```
public function preNuevo($objDatos)
{
   $secuencia = $this->calcularSecuencia('tinv_tipo_bajas','cbaja',array());
   $objDatos->setValue('codigoBaja',$secuencia);
   return 0;
}
```

- **Modificar**

  Por defecto, en gvHidra, se realiza la actualización de todas las tuplas que el usuario ha modificado en el panel. Con los métodos abstractos **preModificar** y **postModificar** podremos modificar este comportamiento.

  Ejemplo Comprobaremos en *preModificar* si la factura tiene fecha de certificación, en cuyo caso no se puede modificar, notificándose al usuario con un mensaje. (Nota: por diseño, en este caso solo se permite la modificación de una factura cada vez, $m_datos solo contiene una tupla)

```
public function preModificar($objDatos)
{
    $indice = key($m_datos);
    if ($objDatos->getValue('fcertificacion')!="")
    {
       $this->showMensaje('APL-12');
       return -1;
    }
    return 0;
}
```

Otro de los ejemplos puede ser el caso de que tengamos que comprobar el estado de una variable del maestro (utilizaremos la clase IgepSession) antes de operar en el detalle. En el ejemplo comprobamos el valor de la variable certificada del maestro para saber si se pueden modificar las líneas de una factura. Si es así, forzamos a que la descripción de las líneas se almacenen en mayúsculas.

```
public function preModificar($objDatos)
{
    //Comprobamos si está la factura certificada... en este caso cancelamos la operacion
    $fechaCertif = IgepSession::dameCampoTuplaSeleccionada('TinvEntradas2','fcertificacion');
    if ($fechaCertif!="")
    {
       $this->showMensaje('APL-12');
       return -1;
    }
    $m_datos = $objDatos->getAllTuplas();
    foreach($m_datos as $indice => $tupla)
    {
        $m_datos[$indice]["dlinea"] = strtoupper($tupla["dlinea"]);
    }
    $objDatos->setAllTuplas($m_datos);
    return 0;
```

```
}//Fin de preModificar
```

El *postInsertar* se ha utilizado para mandar un correo después de la inserción del registro en la tabla:

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $indice = key($m_datos);
  if ($objDatos->getValue('dgral')!='' && $objDatos->getValue('cserv')!=''))
        $nombreServDgral = $conexionOracle->consultar("select ddg as \"nombreDgral\",dservc as \"nombreServicio\"
                                                       from vcmn_organoso where cdgo='".$objDatos->getValue('dgral')."'
                                                       and cservo='".$objDatos->getValue('cserv')."'");

  //Obtenemos los datos de la clave informática,el número reset y el nombre del expediente
  if ($objDatos->getValue('codexp')!='')
  {
        $claveinf = $conexionOracle->consultar("select numero_reset as \"numreset\",cianyo as \"cianyo\",cidg as \"cidg\",
                                               cinum as \"cinum\",citipo as \"citipo\",cinumtipo as \"cinumtipo\",
                                               objeto as \"nombre_exp\"
                                               from vopu_todos_exped where claveseg='".$objDatos->getValue('codexp')."'");
        if(count($claveinf)<=0)
        {
                $this->showMensaje('APL-18');
                return -1;
        }
  }

  //Para poder obtener los datos del usuario de la sesión
  $datosusuario=IgepSession::dameDatosUsuario();

  //Formamos el asunto del mensaje:
  $asunto=$objDatos->getValue('aplicacion')."/".$datosusuario['nombre'];
                ...

  return parent::envioCorreo($m_datos,$rol,$indice,$datosusuario,$asunto,$texto);

}//Fin de postInsertar
```

Otro ejemplo de postInsertar puede ser que al insertar en una tabla maestro queramos que se realice una inserción en una tabla detalle dependiente (entidad débil). Tras realizar esto, por ejemplo, podemos decidir salir de la pantalla con un actionForward de salida, indicado para este caso especial.

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $error = $conexionOracle->operar("INSERT INTO tabla2 VALUES (1,2)");
  if($error==-1)
  {
    $this->obj_errorNegocio->setError("APL-15","TinvLineas2.php","postInsertar");
    return -1;
  }
  else
    return $objDatos->getForward('salir');
}//Fin de postInsertar
```

- **Borrar**

  Con los métodos **preBorrar** y **postBorrar** modificaremos el comportamiento de la operación borrado.

  Ejemplo Vamos a simular el comportamiento de un borrado en cascada. Es decir, si se borra una tupla de la tabla “maestra”, se borrarán todas las tuplas de la tabla “detalle”. En este caso se borran todas las líneas de una factura antes de que se borre la factura.

```
public function preBorrar($objDatos)
{
    $this->operar("DELETE FROM tinv_bienes WHERE
        anyo ='".$objDatos->getValue("anyo")."' and nfactura=".$objDatos->getValue("nfactura")." ");
    $errores = $this->obj_errorNegocio->hayError();
    if($errores)
    {
        $this->showMensaje($this->obj_errorNegocio->getError());
        $this->obj_errorNegocio->limpiarError();
        return -1;
    }
    return 0;
}//Fin de preBorrar
```

Otro de los ejemplos puede ser el caso de que tengamos que comprobar el estado de una variable del maestro (utilizaremos la clase IgepSession) antes de operar en el detalle. En el ejemplo comprobamos el valor de la variable certificada del maestro para saber si se pueden modificar las líneas de una factura. Si es así, forzamos a que la descripción de las líneas se almacenen en mayúsculas.

```
public function preModificar($objDatos)
```

```
{
    //Comprobamos si está la factura certificada... en este caso cancelamos la operacion
    $fechaCertif = IgepSession::dameCampoTuplaSeleccionada('TinvEntradas2','fcertificacion');
    if ($fechaCertif!="")
    {
       $this->showMensaje('APL-12');
       return -1;
    }
    $m_datos = $objDatos->getAllTuplas();
    foreach($m_datos as $indice => $tupla)
    {
        $m_datos[$indice]["dlinea"] = strtoupper($tupla["dlinea"]);
    }
    $objDatos->setAllTuplas($m_datos);
    return 0;
}//Fin de preModificar
```

El *postInsertar* se ha utilizado para mandar un correo después de la inserción del registro en la tabla:

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $indice = key($m_datos);
  if ($objDatos->getValue('dgral')!='' && $objDatos->getValue('cserv')!=''))
        $nombreServDgral = $conexionOracle->consultar("select ddg as \"nombreDgral\",dservc as \"nombreServicio\"
                                                       from vcmn_organoso where cdgo='".$objDatos->getValue('dgral')."'
                                                       and cservo='".$objDatos->getValue('cserv')."'");

  //Obtenemos los datos de la clave informática,el número reset y el nombre del expediente
  if ($objDatos->getValue('codexp')!='')
  {
        $claveinf = $conexionOracle->consultar("select numero_reset as \"numreset\",cianyo as \"cianyo\",cidg as \"cidg\",
                                               cinum as \"cinum\",citipo as \"citipo\",cinumtipo as \"cinumtipo\",
                                               objeto as \"nombre_exp\"
                                               from vopu_todos_exped where claveseg='".$objDatos->getValue('codexp')."'");
        if(count($claveinf)<=0)
        {
                $this->showMensaje('APL-18');
                return -1;
        }
  }

  //Para poder obtener los datos del usuario de la sesión
  $datosusuario=IgepSession::dameDatosUsuario();

  //Formamos el asunto del mensaje:
  $asunto=$objDatos->getValue('aplicacion')."/".$datosusuario['nombre'];
                ...

  return parent::envioCorreo($m_datos,$rol,$indice,$datosusuario,$asunto,$texto);

}//Fin de postInsertar
```

Otro ejemplo de postInsertar puede ser que al insertar en una tabla maestro queramos que se realice una inserción en una tabla detalle dependiente (entidad débil). Tras realizar esto, por ejemplo, podemos decidir salir de la pantalla con un actionForward de salida, indicado para este caso especial.

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $error = $conexionOracle->operar("INSERT INTO tabla2 VALUES (1,2)");
  if($error==-1)
  {
    $this->obj_errorNegocio->setError("APL-15","TinvLineas2.php","postInsertar");
    return -1;
  }
  else
    return $objDatos->getForward('salir');
}//Fin de postInsertar
```

- **Editar**

  La operación de edición se lanza cuando se pasa de modo tabular a modo ficha en un panel mediante la acción “modificar”. Para ello se dispone de dos métodos: **preEditar** y **postEditar**. El método *preEditar* recibe como parámetro de entrada las tuplas que se han seleccionado para pasar al modo ficha. Y el método *postEditar* recibe como parámetro de entrada el resultado de la consulta realizada, de modo que se le pueden añadir tuplas o campos.

  Ejemplo Vamos a comprobar que sólo se pueda modificar un expediente si su estado es “completado”.

```
public function preEditar($objDatos)
{
   while($tupla = $objDatos->fetchTupla())
   {
      if($tupla['estado']!='completado')
      {
          $this->showMensaje('APL-14');
          return -1;
      }
```

```
   }//Fin de foreach
   return 0;
}
```

Otro ejemplo donde vamos a crear un campo de tipo lista para que el usuario pueda introducirlo. El campo unidadesBaja lo tiene que crear el programador e incluirlo en el resultado de la consulta para cada uno de los registros que se hayan seleccionado.

```
public function postEditar($objDatos)
{
   //Cargamos una lista para cada una de las tuplas con los bienes que puede dar de baja
   $m_datos = $objDatos->getAllTuplas();
   foreach($m_datos as $indice => $linea)
   {
        $listaBajas = new IgepLista('unidadesBaja');
        $i=1;
        while($i<=$m_datos[$indice]['unidadesDisp'])
        {
                $listaBajas->addOpcion("$i","$i");
                $i++;
        }
        $m_datos[$indice]['unidadesBaja'] = $listaBajas->construyeLista();
   }//Fin de foreach
   $objDatos->setAllTuplas($m_datos);
   return 0;
}
```

El *postInsertar* se ha utilizado para mandar un correo después de la inserción del registro en la tabla:

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $indice = key($m_datos);
  if ($objDatos->getValue('dgral')!='' && $objDatos->getValue('cserv')!=''))
        $nombreServDgral = $conexionOracle->consultar("select ddg as \"nombreDgral\",dservc as \"nombreServicio\"
                                                       from vcmn_organoso where cdgo='".$objDatos->getValue('dgral')."'
                                                       and cservo='".$objDatos->getValue('cserv')."'");

  //Obtenemos los datos de la clave informática,el número reset y el nombre del expediente
  if ($objDatos->getValue('codexp')!='')
  {
        $claveinf = $conexionOracle->consultar("select numero_reset as \"numreset\",cianyo as \"cianyo\",cidg as \"cidg\",
                                               cinum as \"cinum\",citipo as \"citipo\",cinumtipo as \"cinumtipo\",
                                               objeto as \"nombre_exp\"
                                               from vopu_todos_exped where claveseg='".$objDatos->getValue('codexp')."'");
        if(count($claveinf)<=0)
        {
                $this->showMensaje('APL-18');
                return -1;
        }
  }

  //Para poder obtener los datos del usuario de la sesión
  $datosusuario=IgepSession::dameDatosUsuario();

  //Formamos el asunto del mensaje:
  $asunto=$objDatos->getValue('aplicacion')."/".$datosusuario['nombre'];
                ...

  return parent::envioCorreo($m_datos,$rol,$indice,$datosusuario,$asunto,$texto);

}//Fin de postInsertar
```

Otro ejemplo de postInsertar puede ser que al insertar en una tabla maestro queramos que se realice una inserción en una tabla detalle dependiente (entidad débil). Tras realizar esto, por ejemplo, podemos decidir salir de la pantalla con un actionForward de salida, indicado para este caso especial.

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $error = $conexionOracle->operar("INSERT INTO tabla2 VALUES (1,2)");
  if($error==-1)
  {
    $this->obj_errorNegocio->setError("APL-15","TinvLineas2.php","postInsertar");
    return -1;
  }
  else
    return $objDatos->getForward('salir');
}//Fin de postInsertar
```

- **Recargar**

  Esta acción se realiza cuando vamos a mostrar la información de un panel que depende de otro, es decir, cuando vamos a mostrar la información de un detalle en un maestro-detalle. Se proporcionan dos métodos para parametrizar esta acción: **preRecargar** y **postRecargar**. El primero recibe como parámetro la selección del maestro, y el segundo el resultado de la consulta.

Ejemplo Vamos a mostrar como recargar un detalle que no esté compuesto de una select. En este caso, cargamos el obj_ultimaConsulta (objeto que contiene lo que se va a mostrar en pantalla) con una matriz de datos que tenemos previamente cargada en la clase. Al tratarse de un detalle, esta matriz de datos dependerá de la selección realizada en el padre. En nuestro ejemplo el campo clave es ccentro, y la estructura de la variable líneas es una matriz del modo [ccentro][indice][campo]. De modo que al cambiar el ccentro, sólo mostraremos las líneas que correspondan a dicho centro.

```
public function postRecargar($objDatos)
{
   $ccentroSeleccionado = $objDatos->getCampo('ccentro','seleccionarPadre');
   $lineas = IgepSession::dameVariable('InvCabCertificadosBaja','lineasCertificados');
   $objDatos->setAllTuplas($lineas[$ccentroSeleccionado]);
   return 0;
}
```

- **Buscar**

  El método **preBuscar** se dispara antes de que se realice la consulta de búsqueda (la que viene del modo búsqueda). Recibe como parámetro un objeto *IgepComunicaUsuario*, que permite al programador navegar a través de los datos que el usuario ha introducido en el modo de búsqueda. Así, y haciendo uso de métodos como *setSearchParameters*, el programador puede añadir restricciones a la consulta a realizar como EXISTS o 'fecha BETWEEN fechaInicio AND fechaFin'

  El método *postBuscar* recibe el mismo parámetro pero con el resultado de la consulta realizada, de modo que se puede modificar el contenido de dicha matriz de datos añadiendo y/o quitando registros y/o campos.

  Los métodos relativos a la búsqueda están comentados en profundidad en la sección del uso del panel de búsqueda.

- **Abrir y cerrar aplicación**

  El framework ofrece la posibilidad de ejecutar código antes de abrir y cerrar una ventana. Para ello, tendremos que hacer uso de una clase manejadora especial que se encargará de controlar el panel principal (pantalla de entrada). En esta clase, podemos sobreescribir los métodos **openApp** y **closeApp**, y modificar el comportamiento.

  Ejemplo Mostraremos como añadir un mensaje al iniciar la aplicación (método *openApp*)

```
class AppMainWindow extends CustomMainWindow
{
   public function AppMainWindow()
   {
        parent::__construct();
        //Cargamos propiedades específicas del CS
        //Cuando estamos en desarrollo registramos todos los movimientos
        $conf = ConfigFramework::getConfig();
        $conf->setLogStatus(LOG_NONE);
        $conf->setQueryMode(2);
    }

    public function openApp()
    {
       $this->showMensaje('APL-31');
       return 0;
    }
} //Fin de AppMainWindow
```

3.5.2. Uso del panel de búsqueda

3.5.2.1. ¿Qué hace la acción buscar?

Buscar es la encargada de construir la consulta (SELECT) del panel utilizando los campos del panel para construir la WHERE, creando una cadena del tipo campo=valor si estos campos no son vacíos, para que esta cadena se cree es necesario que los campos del panel de búsqueda estén definidos con el método **addMatching()** en la clase de negocio correspondiente al panel. Los métodos para realizar esta búsqueda son: setSelectForSearchQuery(), setWhereForSearchQuery() y setOrderByForSearchQuery() [39].

La acción de buscar, tal y como se ha comentado en el punto anterior Operaciones y métodos virtuales [64], el programador dispone de dos métodos que puede sobrescribir para parametrizar el comportamiento de la búsqueda, **preBuscar()** y **postBuscar()**. El primero nos puede servir para modificar la SELECT antes de lanzarla (añadir constantes, añadir un filtro a la WHERE,...) o para impedir la realización de la consulta. El segundo puede servir, por ejemplo, para añadir columnas adicionales al DBResult obtenido.

3.5.2.2. ¿Qué tipos de búsqueda realiza internamente gvHidra? ¿Cómo puedo cambiarlos?

Tal y como se ha comentado en el apartado anterior, gvHidra crea una cadena que añade al WHERE de la consulta del panel con los datos que ha introducido el usuario desde el modo de búsqueda. Esta cadena se puede formar de tres formas diferentes, dependiendo del valor que se le pase al método **setTipoConsulta()**, teniendo en cuenta que en ninguna de las opciones se distingue por mayúsculas/minúsculas ni por acentos u otros carácteres especiales:

- *Valor 0*: Se construye la WHERE igualando los campos a los valores.

  Quedaría como: *campo = valor*.

- *Valor 1*: Se construye la WHERE utilizando el operador LIKE.

  Quedaría como: *campo LIKE %valor%*

- *Valor 2*: Se construye la WHERE igualando los campos a los valores siempre y cuando estos no contengan el carácter % o _ , en cuyo caso se utilizará el LIKE.

NOTA: gvHidra tiene marcado por defecto el **valor 2** para ejecutar las consultas.

El programador fija el tipo de consulta general para toda la aplicación mediante la propiedad **queryMode** que se encuentra en los ficheros de configuración, gvHidraConfig.xml. Si surge la necesidad de modificar este comportamiento para un panel se puede hacer uso del método **setTipoConsulta()** en la clase correspondiente. En las ventanas de selección también se puede configurar la búsqueda de forma similar con el método **setQueryMode()**.

3.5.2.3. ¿Cómo inicializamos un panel sin pasar por la búsqueda?

Se puede dar el caso de que el programador quiera visualizar directamente los datos sin pasar por el panel de búsqueda previamente. Esto se puede solucionar dándole en la entrada de menu de la ventana (menuModulos.xml) la url: **phrame.php?action=ClaseManejadora__buscar**.

Con esto, se realizará la busqueda de forma automática cargando los datos sin filtros extra.

```
<opcion titulo="Estados" descripcion="Mantenimiento de Estados" url="phrame.php?action=TinvEstados__buscar"/>
```

3.5.2.4. ¿Cómo parametrizar la consulta?

Puede ser que surja la necesidad de parametrizar la búsqueda antes de lanzarla. Con este propósito, tenemos el método **setParametrosBusqueda()**, que se incluirá en la pre-búsqueda, para indicarle algunos valores que bien, filtren la SELECT del panel, o que queremos que nos aparezcan. Puede darse el caso de que el filtro a introducir corresponda a un campo que no tiene matching o no sea del tipo campo=valor (is null, EXISTS,...). Por ejemplo: imaginemos que tenemos un panel de búsqueda sobre datos relativos a facturas en el que hemos introducido una lista desplegable llamada abonada que tiene 3 opciones: "Si", "No" y "". Puede darse el caso que en los datos, para saber si una factura está abonada o no, tengamos que comprobar si tiene valor el campo "fechaAbono". ¿Como hacemos esto? Utilizando el método setParametrosBusqueda antes de realizar la búsqueda.

```
function preBuscar($objDatos)
{
    $abonada = $objDatos->getValue('abonada')
    if($abonada!='')
```

```
    {
        if($abonada=='Si')
            $where = 'fechaAbono is not NULL';
        elseif($abonada=='No')
            $where = 'fechaAbono is NULL';
        $this->setParametrosBusqueda($where);
    }
    return 0;
}
```

También podemos añadir condiciones sobre campos de otras tablas relacionadas. Con el método **unDiacriticCondition()** obtenemos una condición respetando el **queryMode** del formulario (aunque descartando carácteres especiales y mayúsculas):

```
function preBuscar($objDatos)
{
    $con = $this->getConnection();
    $desc = $objDatos->getValue('descrip_linea');
    if($desc!='')
    {
        $condic_lineas = $con->unDiacriticCondition('descrip_linea',$desc, $this->getTipoConsulta());
        $where = 'id_factura in (select id_factura from lineas where '.$condic_lineas.')';
        $this->setParametrosBusqueda($where);
    }
    return 0;
}
```

3.5.2.5. ¿Cómo limitar los registros mostrados?

gvHidra por defecto coloca un límite de 100 registros a todas la consultas realizas a partir de la acción buscar, pero esto se puede cambiar a gusto del programador para cada uno de los paneles. Puede darse el caso que un panel tenga una SELECT muy compleja y se quiera limitar el número de registros mostrados a 20. Esto se puede hacer invocando la función de negocio **setLimiteConsulta()**.

```
$this->setLimiteConsulta(25);
```

3.5.2.6. ¿Cómo añadir constantes a una consulta?

Siguiendo con el ejemplo anterior, puede resultar interesante cargar constantes antes de ejecutar la SELECT. Para ello tenemos el método **addConstante()** que se encarga de añadir la constante en la Select para que aparezca en el DBResult.

```
function preBuscar($objDatos)
{
  $actividad = IgepSession::dameVariable('TinvEstados','Actividad');
  $this->addConstante('Prueba',$actividad);
  return 0;
}
```

3.5.2.7. ¿Cómo conseguir que los campos que componen el filtro mantengan el valor tras pulsar buscar?

Es bastante útil para un usuario que, tras buscar, pueda mantener en el modo filtro los valores introducidos en dicho filtro. Esto les puede saber para, por ejemplo, saber por que año han filtrado. gvHidra permite que, activando un flag, se recuerden automáticamente los valores introducidos. Para activar dicho flag (por defecto viene desactivado) debe utilizar en la clase manejadora el método **keepFilterValuesAfterSearch()**.

```
public function __construct() {
  ...
  $this->keepFilterValuesAfterSearch(true);
  ...
}
```

Para hacer uso de esta utilidad, se recomienda distinguir entre los campos del filtro y el listado/edición.

3.5.2.8. ¿Cómo parametrizar el comportamiento después de la inserción?

gvHidra, por defecto, después de realizar una inserción en el modo listado (searchMode), cambia el filtro para mostrar únicamente los nuevos registros insertados, si lo que queremos es volver a realizar la búsqueda anterior se debe invocar al método **showOnlyNewRecordsAfterInsert()** con el parámetro false.

```
public function __construct()
{
    ...
    $this->showOnlyNewRecordsAfterInsert(false);
    ...
}
```

3.5.2.9. ¿Cómo cambiar los filtros que crea el FW?

En gvHidra, al realizar una de las acciones de consulta (buscar/editar), construye una WHERE y la almacena para poder refrescar tras las operaciones de modificación. Puede que se de el caso, en el que necesitemos cambiar esos filtros para poder visualizar unos registros en concreto (p.e. en un acción particular). Para estos casos, podemos acceder/cambiar dichos filtros con los siguientes métodos:

- getFilterForSearch(): Obtiene el valor del filtro que se ha construido sobre la SearchQuery tras la acción de buscar.
- setFilterForSearch(): Cambia el valor del filtro que actúa sobre SearchQuery.
- getFilterForEdit(): Obtiene el valor del filtro que se ha construido sobre la EditQuery tras la acción de editar.
- setFilterForEdit(): Cambia el valor del filtro que actúa sobre EditQuery.

```
//Cambia el filtro para SearchQuery
$this->setFilterForSearch("WHERE id=1");
```

Nota: si estamos en una acción de interfaz, será necesario recargar la consulta bien con refreshSearch() o refreshEdit().

3.5.3. Acciones no genéricas

Vamos a explicar ahora como podemos incorporar en una ventana acciones no genéricas.

Es bastante común que tengamos ventanas en una aplicación que tengan un comportamiento no genérico, es decir, que no podamos resolverlas mediante una acción predefinida. Los casos con los que nos podemos encontrar son:

1. Generación de listados.
2. Mostrar ventanas emergentes.
3. procesos complejos o poco comunes (enviar un correo, procesos de actualización complejos...)

Para estos casos se ha incorporado un método virtual en las clases manejadoras que cede el control de la ejecución al programador para que pueda realizar las operaciones que estime oportuno.

3.5.3.1. Acceso a datos

Antes de explicar como podemos incorporar este tipo de acciones al framework, tenemos que hacer una referencia importante al acceso a los datos. Como ya sabréis, el framework tiene una serie de métodos de extensión para cada una de las acciones que permiten al programador acceder a los datos y modificarlos. Estos datos a los que se accede siempre van vinculados a la acción que se ejecuta. Por ejemplo para una acción insertar obtendremos los datos que se quieren utilizar para construir la INSERT.

Ahora bien, ¿qué datos vamos a utilizar en una acción particular? Esto dependerá de lo que queramos hacer. Puede que queramos recoger los datos que el usuario haya marcado para borrado para realizar un borrado; puede que queramos los datos que el usuario acaba de introducir como nuevos,...

Por todo ello, debemos indicar al framework el conjunto de datos con el que queremos trabajar, para ello debemos utilizar (y es obligatorio para este tipo de acciones) el método **setOperacion()** de la instancia de IgepComunicaUsuario proporcionada. Este método admite como parámetro un string que es el nombre de la operación sobre la que queremos trabajar. Algunas de las más importantes son:

- **insertar**: Los datos que el usuario ha introducido para insertar.
- **actualizar/modificar**: Los datos que el usuario ha introducido para modificar.
- **borrar**: Los datos que el usuario ha introducido para borrar.
- **seleccionar**: Los datos que el usuario ha seleccionado.
- **visibles**: Los datos que el usuario tiene visibles en pantalla.
- **buscar**: Los datos que el usuario ha introducido en el panel de búsqueda.
- **external**: Los valores de los campos external (ver campos especiales).

Cambiando entre las operaciones podremos acceder a los datos correspondientes.

3.5.3.2. Implementación de una acción particular

A continuación iremos relatando paso a paso como añadir acciones particulares a una ventana.

En primer lugar tenemos que añadir un estímulo en la pantalla que lance dicha acción, típicamente un botón. Para ello debemos añadir en la tpl, un botón que lance la acción. Lo indicaremos con los parámetros **accion** y **action**:

```
{CWBoton imagen="50" texto="Generar" class="boton" accion="particular" action="nombreDeTuOperacion"}
```

Una vez creado el disparador en la pantalla, debemos indicar que clase debe gestionar la acción y que posibles destinos tendrá de respuesta. Para ello, en el fichero *mappings.php* añadimos la siguiente información:

```
$this->_AddMapping('claseManejadora__nombreDeTuOperacion', 'claseManejadora');
$this->_AddForward('claseManejadora__nombreDeTuOperacion', 'operacionOk',
       'index.php?view=views/ubicacioFicheroViewsClaseManejadora.php&panel=buscar');
$this->_AddForward('claseManejadora__nombreDeTuOperacion', 'operacionError',
       'index.php?view=views/ubicacioFicheroViewsClaseManejadora.php&panel=buscar');
```

Finalmente, en la clase manejadora debemos extender el método **accionesParticulares()** para poder recibir el control de la ejecución y realizar las operaciones oportunas. El código sería:

```
class claseManejadora extends gvHidraForm
{

...

    public function accionesParticulares($str_accion, $objDatos)
    {

        switch($str_accion)
        {
            case "nombreDeTuOperacion":

                // En el ejemplo usamos:
                // -comunicacion de errores mediante excepcion.
                // -recojida de datos seleccionados en pantalla.
                // -creacion de instancias de clases de negocio (podrian
ser llamadas estaticas).
                try {
                    //Recojo valores de las tuplas seleccionadas
                    $objDatos->setOperacion('seleccionar');
                    $datos = $objDatos->getAllTuplas();

                    //Llamamos a la clase de negocio
                    $f = new funciones();
                    $f->miMetodo($datos);
                    $actionForward =
$objDatos->getForward('operacionOk');
                }
                catch(Exception $e)
                {
                   //Mostramos mensaje de error
                   $this->showMensaje('APL-01');
                   $actionForward =
$objDatos->getForward('operacionError');
                }
                break;

            case "nombreDeTuOperacion2":
                ...
                break;
```

```
        }
        return $actionForward;
   }
```

Antes de una acción particular SIEMPRE tenemos que indicar el tipo de datos a los que queremos acceder, lo haremos con **setOperacion()**. Recordemos que tenemos diferentes operaciones (insertar, modificar, borrar, seleccionar, visibles, external...)

El método **accionesParticulares()** debe devolver un objeto **actionForward** válido. Tenemos acceso a los forwards de nuestra acción a través del método **getForward** del objeto de datos (en el ejemplo $objDatos).

3.5.3.3. Ejemplo de listado

Un caso típico, es el de los listados. El procedimiento es el mismo al explicado anteriormente, pero debemos añadir a la generación de listado la posibilidad de generarse en una ventana emergente. Para ello debemos hacer uso del método **openWindow** pasándole el forward de destino. Por ejemplo:

```
class claseManejadora extends gvHidraForm
{
  ...

   public function accionesParticulares($str_accion, $objDatos)
   {
        ...
        //Si todo está Ok
        //Abrimos una nueva ventana donde mostraremos el listado
        $actionForward = $objDatos->getForward('mostrarListado');
        $this->openWindow($actionForward);

        //En la ventana actual mostramos un mensaje indicando que el listado se ha generado
        //y volvemos a la ruta deseada
        $this->showMensaje('APL-29');
        $actionForward = $objDatos->getForward('operacionOk');
    }
}
```

3.5.4. Acciones de interfaz

3.5.4.1. Definición

Las acciones de interfaz en gvHidra es todo aquel estímulo de pantalla que se resuelve con un cambio de la interfaz sin recarga de la ventana. Es decir, típicamente el uso de tecnología AJAX para actualizar la interfaz sin que el usuario perciba una recarga de la página. Algunos ejemplos de acciones que podemos necesitar son:

- Cuando pierda el foco un campo, actualizar un campo descripción (típica descripción de claves ajenas)
- Dependiendo del valor de un desplegable (HTML select), mostrar/ocultar un campo en pantalla.
- Cuando marcamos/desmarcamos un checkbox, habilitar/deshabilitar un campo.
- ...

Para ello, el framework intenta simplificar el uso de estas acciones de forma que el programador centrará sus esfuerzos en la clase manejadora (en PHP); dejando que la herramienta se encargue de la generación de Javascript. Actualmente el framework utiliza un iframe oculto que recoge el estímulo de pantalla, lo direcciona a la clase manejadora correspondiente y recibe la respuesta en formato Javascript.

3.5.4.2. Uso de las acciones de interfaz

El primer paso para definir una acción de interfaz es indicar en la TPL que el campo tiene que disparar dicha acción.

En la tpl hemos incluido el siguiente código:

```
{CWCampoTexto nombre="cif" editable="true" size="13" textoAsociado="Cif"}
```

```
 
{CWCampoTexto nombre="orden" editable="true" size="2" textoAsociado="Orden" actualizaA="nombre"}
 {CWCampoTexto nombre="nombre" editable="false" size="40"}
```

Se ha añadido la propiedad **actualizaA.** Esta propiedad le indica a la clase que cuando pierda el foco el campo *orden* se tiene que lanzar una acción de interfaz.

Ahora, en la clase, debemos indicar que *acción de interfaz* corresponderá con la pérdida de foco de dicho campo. Para ello incluimos en el constructor de la clase manejadora el siguiente código:

```
//Para las validaciones cuando introduzca el proveedor
$this->addAccionInterfaz('orden','manejadorProveedor');
```

En el método se introduce el nombre del campo de la tpl que lanzará la validación y el nombre del método que el programador va a implementar para realizar la validación.

La implementación del método se hace en la clase asociada al panel:

```
public function manejadorProveedor ($objDatos) //Manejador del evento
{
  $cif = $objDatos->getCampo('cif');
  $orden = $objDatos->getCampo('orden');

  if($cif!='' and $orden!='')
  {
     if (!$this->validaCif($cif))//Validación léxica
     {
        $this->showMensaje('APL-04',array($cif));
        return -1
     }

     $proveedor = $this->obtenerProveedor($cif, $orden);
     if ($proveedor == null) // Validación semántica
     {
        $this->showMensaje('APL-24',array($cif,$orden));
        return -1
     }
     else
     {
        $objDatos->setValue('nombre',$proveedor);
     }
  }
  else
     $objDatos->setValue('nombre','');
  return 0;
}

public function validarCif ($cif, $orden)
{
...
}

public function obtenerProveedor($cif, $orden)
{
  $res = $this->consultar("select * from tcom_proveed where nif = '".$cif."' and orden = ".$orden );
  if(count($res)>0)
  {
     return($res[0]['nombre']);
  }
  else
  {
     return(null);
  }
}
```

En resumen, el ejemplo anterior realizará una comprobación de los campos cif y orden, una vez rellenados por el usuario (concretamente cuando pierda el foco el campo orden), lanzando el método *manejadorProveedor()* y tendrá el resultado si el cif y el orden es correcto. En caso contrario, el resultado será alguno de los mensajes de error que el programador haya capturado.

En general, la interfaz y el uso de estos métodos es parecida a la de cualquier método pre o post operación. Si devuelven 0 se considera que todo ha ido correctamente y si devuelve -1 que se ha encontrado un error.

Sin embargo, estos métodos tienen una peculiaridad especial, además de las funcionalidades de las que dispone un método pre o post operación (como sabemos, se permite el acceso a los datos que hay en pantalla mediante la variable $objDatos, se pueden realizar consultas sobre la fuente de datos del panel, ...) pueden modificar aspectos de estado de la interfaz como el contenido, la visibilidad o la accesibilidad de los componentes de pantalla. Para ello debemos hacer uso de los siguientes métodos:

- **getValue($campo)**: Obtiene el valor de un campo

- **setValue($campo,$valor)**: Cambia el valor de un campo
- **setVisible($campo,$valor)**: Cambia la visibilidad de un campo
- **setSelected($campo,$valor)**: Cambia el valor seleccionado de una lista.
- **setEnable($campo,$valor)**: Cambia la accesibilidad de un campo.
- **posicionarEnFicha($indiceFila)**: Nos permite cambiar la ficha activa.
- **getModoActivo()**: Devuelve el modo que dispara la acción.
- **getCampoDisparador()**: Devuelve el campo que lanza la acción de interfaz.
- **setBttlState($idPanel, $nameBttl, $on)**: Habilita o deshabilita el botonTooltip básico ('insertar', 'modificar', 'restaurar') del panel indicado.

Ejemplo:

```
//Fijar la visibilidad de un campo
$objDatos->setVisible('nfactura',true);
$objDatos->setVisible('nfactura',false);
//Fija la accesibilidad de un campo
$objDatos->setEnable('nfactura',true);
$objDatos->setEnable('nfactura',false);
//Fija el contenido de un campo
$objDatos->getValue('nfactura');
$objDatos->setValue('nfactura','20');

//Deshabilita el boton Tooltip de inserción:
$objDatos->setBttlState('edi', 'insertar', false);

//Obtienen información sobre el modo que dispara la acción (FIL-LIS-EDI)
$objDatos->getModoActivo();
```

3.6. Personalizando el estilo

El framework suministra un método para personalizar su aspecto y funcionamiento, que en adelante llamaremos 'temas'. Los temas están ubicados en la carpeta en **igep/custom**, podemos tener varios temas y activar uno en concreto mediante el atributo **customDirName** del fichero gvHidraConfig.inc.xml del framework o de la aplicación, o con el método **ConfigFramework->setCustomDirName** (ver configuración de gvHidraConfig.xml [30]).

Con gvHidra vienen tres temas configurados:

- **cit.gva.es**: es el usado internamente en la Conselleria de Infraestructuras y Transportes. Es el más completo, ya que además del estilo tambien incluye funcionalidad.
- **default**: es el mismo aspecto que el anterior pero sin funcionalidad extra ni características propias de la Conselleria.
- **gvpontis**: es el usado en la web gvpontis [http://www.gvpontis.gva.es/], y tampoco incluye funcionalidad extra.

Vamos a pasar a explicar la estructura inicial que debe contener un tema.

3.6.1. CSS

Tendremos un directorio CSS donde tendremos la hoja de estilo a utilizar en nuestra aplicación (*aplicacion.css*) y otra hoja de estilo para el menú desplegable (*layersmenu-cit.css*). La hoja de estilo que se aplica al calendario en estos momentos no es personalizable.

Para personalizar la hoja de estilo aplicacion.css entendiendo las reglas de estilo que se han definido para gvHidra existe una explicación detallada de qué significa cada una y dónde se aplica cada selector en el punto Creación de un custom propio del capítulo 7 [139].

3.6.2. Imágenes

En el directorio imágenes tenemos una estructura de carpetas que hay que respetar, a la vez que también hay que respetar los nombres de los fichero, ya que el framework buscará en esa ruta y con ese nombre.

Explicación de la estructura de directorios:

- **acciones**: Imágenes para los CWBoton y las flechas de ordenación en un CWTabla (14.gif y 16.gif).
- **arbol**: Imágenes que se utilizan cuando trabajemos con el patrón árbol.
- **avisos**: Imágenes que se utilizan para los diferentes mensajes que utiliza el framework (avisos, sugerencias, errores...)
- **botones**: Imágenes de los botones tooltip. Existen dos imágenes por cada botón, uno con la imagen en color para cuando el botón está activo y otra para el estado inactivo, estos llevan el sufijo "off".
- **listados**: Logo para listados.
- **logos**: Imágenes para los logos de la pantalla principal y menú desplegable de la barra superior.

  "logo.gif": Logo que aparece al final de la pantalla de entrada.

  "logoMenu.gif": Logo del menú de la barra superior de las pantallas.

  "logoSuperior.gif": Logo de fondo en la pantalla de entrada.
- **menu**: Imágenes para los menús de la pantalla principal.
- **paginacion**: Imágenes para los botones de la paginación.
- **pestanyas**: Imágenes para las pestañas de los modos de trabajo.
- **tablas**: Imágenes que se utilizan en tablas. Actualmente tenemos la imagen que nos servirá para marcar la línea de separación de la cabecera en las tablas.

3.6.3. Ficheros de configuración del custom

Tenemos cuatro ficheros que nos permitirán una configuración del custom que sobreescribirá la propia del framework, pero, que a la vez estará por debajo de la configuración de cada aplicación:

- **include.php**: fichero donde se deben incluir las clases propias del custom. Pueden ser clasesManejadoras (directorio actions), tipos (directorio types), clases de negocio (directorio class), ...
- **CustomMapping.php**: define los mappings propios del custom.
- **gvHidraConfig.inc.xml**: define la configuración estática del custom.
- **CustomMainWindow**: define la configuración dinámica del custom.

3.7. Tratamiento de tipos de datos

En un formulario podemos tener campos o columnas de varios tipos de componentes, entre otros campos de texto, combos, checkbox, ... En el caso de los campos de texto, podemos definir su tipo y algunas características más, con lo que el framework nos va a facilitar algunas tareas básicas asociadas a estos. Una vez que hemos definido el tipo, tenemos que enlazarlo con el campo en la template con el parámetro **dataType**.

Ejemplo:

```
# clase manejadora
$this->addFieldType('filTelefono', new gvHidraString(false, 15));

# template
{CWCampoTexto nombre="filTelefono" size="15" editable="true" textoAsociado="Teléfono"
              dataType=$dataType_nombreClaseManejadora.filTelefono}
```

3.7.1. Características generales

El framework proporciona diferentes tipos de datos que permiten controlar varios aspectos de interfaz y comportamientos. Concretamente, al indicar el tipo de datos de un campo el framework nos ofrece:

- Máscaras de entrada según tipo de datos en el cliente.
- Validaciones en el servidor antes de operaciones de modificación de los mismos en la BD (modificación e inserción).
- Limitar la longitud máxima del campo en pantalla, comprobaciones a nivel cliente (maxlength) y servidor
- Comprobación de obligatoriedad a nivel de cliente y servidor.
- Ordenación en tabla por tipo de datos (sino se indica ordena siempre como cadena).
- Mostrar en pantalla características especiales según tipo de datos.

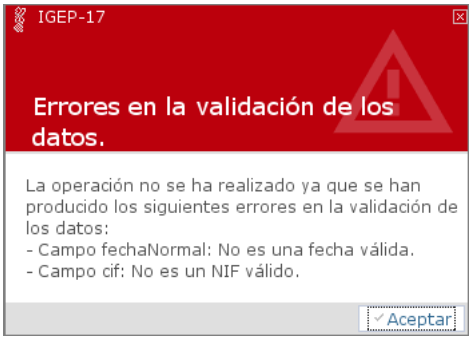
Los tipos de datos disponibles en gvHidra son:

1. **gvHidraString**
2. **gvHidraDate**
3. **gvHidraDateTime**
4. **gvHidraInteger**
5. **gvHidraFloat**

Para hacer uso de los tipos, en la clase manejadora hay que crear una instancia con el tipo que queramos, y luego asignarla al campo con el método **addTypeField()** de la clase manejadora.

Para informar al plugin (CWCampoTexto, CWAreaTexto, CWLista...) el tipo de datos que va a contener, tenemos que utilizar la propiedad **dataType**. El framework insertará los datos en la variable *$dataType_claseManejadora.nombreCampo*.

Con esta definición de tipos conseguimos que el framework valide los datos antes de cada operación de modificación de la BD (actualización e inserción). Al detectar algún error, de forma automática, recogerá la información y la mostrará en pantalla informando al usuario.



3.7.2. Cadenas (gvHidraString)

Para el tratamiento de cadenas tenemos el tipo gvHidraString. Este tipo tiene los siguientes métodos propios:

- **setRegExp(string)**: Permite introducir una expresión regular para validar la cadena en el cliente (cuando el campo pierde el foco) como en el servidor.
- **setInputMask(string)**: Permite introducir una máscara de entrada de datos para la cadena. El formato de máscara será
  - '**#**' Indica un carácter numérico.
  - '**x**' Indica una letra (mayúscula o minúsculas)
  - '*****' Indica un carácter alfanumérico (cualquier letra o número)
  - '**(**', '**)**', '**-**', '**.**', etc: Caracteres delimitadores típicos

Vamos a ver algunos ejemplos para que quede más claro el uso de las máscaras:

- *Máscara*: (##) ########
  - *Cadena*: 963333333
  - *Resultado*: (96) 3333333
- *Máscara*: CP: #####
  - *Cadena*: 46001
  - *Resultado*: CP: 46001

Nota: si no se hace uso de los métodos los mismos están deshabilitados.

En el siguiente fragmento de código vemos como podemos asignar el tipo gvHidraString a un campo.

```
$telefono = new gvHidraString(false, 15);
$telefono->setInputMask('(+##)-#########');
$this->addFieldType('filTelefono',$telefono);
```

Primero se crea un objeto de tipo string llamando a la clase gvHidraString, con el primer parámetro indicamos la obligatoriedad del campo, y con el segundo el tamaño máximo del campo. Ahora ya podemos hacer uso del método **setInputMask()**, que marcará el campo con esa máscara. Y por último se asigna el tipo al campo en concreto, en nuestro caso es "filTelefono" (alias del campo en la select de búsqueda o edición).

3.7.3. Fechas

3.7.3.1. Entendiendo las fechas en gvHidra

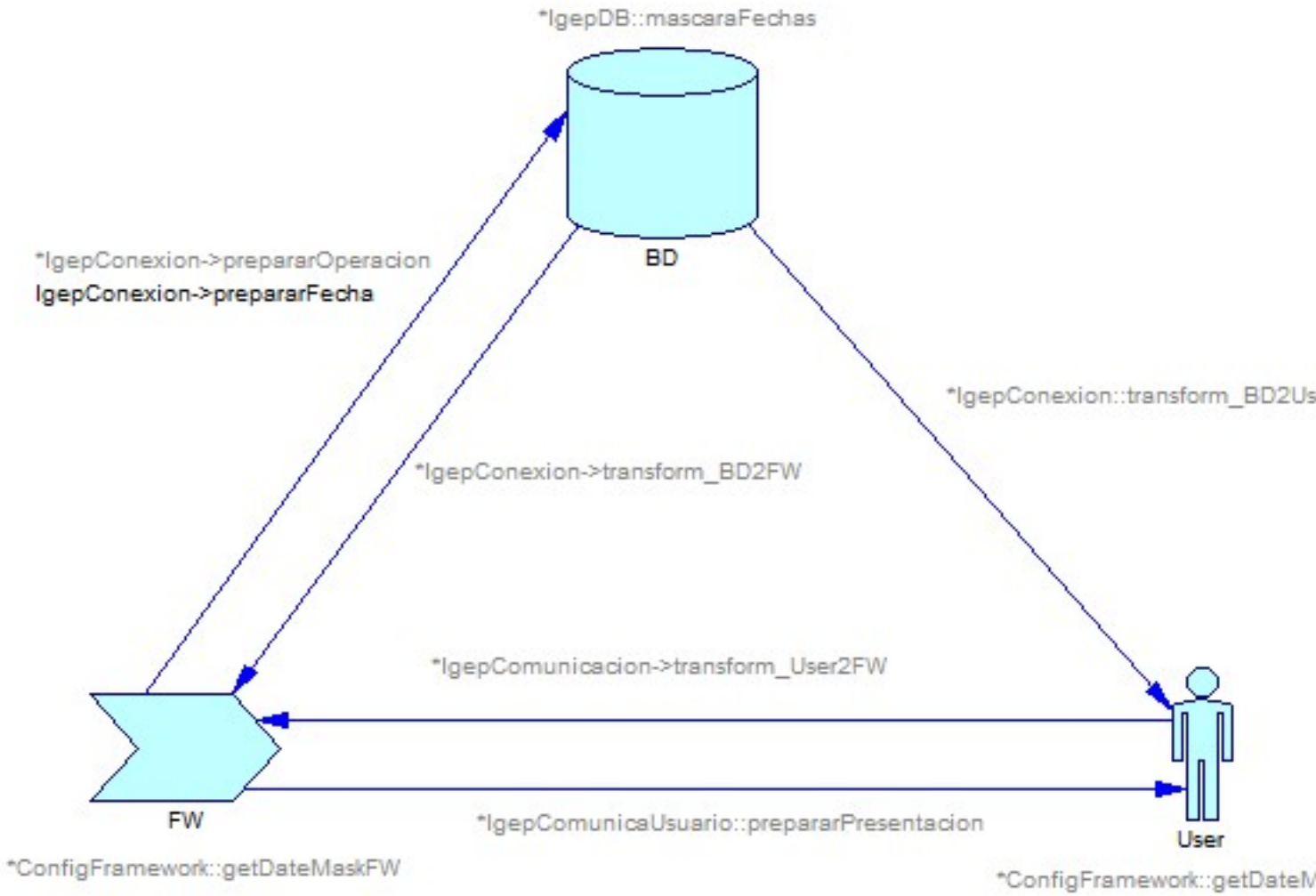
La intención de gvHidra es simplificar lo máximo posible los problemas derivados de los formatos de las fechas, y para ello se ofrece al programador una forma única de manejar las fechas (formato de negocio) y es el framework el que se encarga, en caso necesario, de convertir estas fechas al formato del usuario para interaccionar con éste (formato de usuario), o al formato que entiende nuestro gestor de base de datos para almacenar/recuperar los datos (formato de datos).

Actualmente gvHidra define de manera global el formato con el que va a mostrar las fechas al usuario (método **ConfigFramework::getDateMaskUser**) y se puede cambiar por el programador.

También se define, por cada tipo de SGBD, cual es el formato que reconocen (método **mascaraFechas** de **IgepDBMS_*)** aunque no se recomienda modificarlos.

Por último, también se define el modo de representar las fechas internamente (método **ConfigFramework::getDateMaskFW**), que es el que nos permitirá operar en PHP con las fechas y que tampoco se recomienda modificarlo.

En la siguiente figura podemos ver un esquema de la situación general:



Cada nodo representa uno de los tipos de formato explicados antes, y las etiquetas junto a ellos representan el método usado para obtener la definición de ese formato. Las transiciones entre nodos representan las conversiones que se realizan en el framework, y las etiquetas junto a ellas representan los métodos que convierten del formato origen al destino de la transición. Además los métodos marcados con * (y con color más claro) indican que el método es interno al framework, y por tanto el programador no necesitará usarlo normalmente. A pesar de ello se han incluido todos ellos en el esquema para dar una visión completa.

3.7.3.2. Uso en el framework

Primero, para incluir una fecha en un panel necesitamos:

- Definir el campo en la select (búsqueda o edición).

- Definir el campo equivalente en la plantilla (tpl).
- Asociar un tipo de datos al campo de la plantilla (tpl) a través del constructor de la clase manejadora. Usaremos las clases gvHidraDate o gvHidraDatetime dependiendo de si queremos la hora o no.

```
// Fecha con calendario y etiqueta para el día de la semana en formato corto
// El parámetro indica "obligatoriedad" para cumplimentar el campo
$tipoFecha = new gvHidraDate(true);
// Existirá un calendario asociado al campo
$tipoFecha->setCalendar(true);
//Aparecerá la inicial del día de la semana (L, M, X, J, V, S, D)
tipoFecha->setDayOfWeek('short');
$this->addFieldType('fcertificacion', $tipoFecha);
```

Cuando el programador accede a un campo declarado como fecha (en un método "pre" o "post", en una operación o en una acción particular) obtiene un objeto de la clase gvHidraTimestamp (basada en DateTime [http://php.net/manual/es/book.datetime.php]) (si no tiene valor se recibirá un NULL). No confundir esta clase con las usadas para definir el tipo, ésta es la que se usa para operar con las fechas (con o sin hora).

```
$fpeticionIni = $objDatos->getValue('fpeticionIni');
if (empty($fpeticionIni))
    return;
// en $fpeticionIni tenemos una instancia de gvHidraTimestamp

// incrementar la fecha en 1 dia
$fpeticionIni->addDays(1);

// inicializar un campo a la fecha actual:
$objDatos->setValue('fcreacion', new gvHidraTimestamp());
```

La clase gvHidraTimestamp nos ofrece algunos métodos útiles para modificar u operar sobre las fechas:

- **__construct([$ts='now' [, $tz=null]])**: el constructor tiene los mismos parámetros que DateTime; si no le pasamos ninguno se inicializa a la fecha y hora actual.
- **setTime($hours, $minutes [, $seconds = 0])**: para modificar la hora.
- **setDate($year, $month, $day)**: para modificar la fecha.
- **modify($str)**: método de DateTime que usa la sintaxis de strtotime para modificar la fecha.
- **addDays(int), subDays(int)**: añadir y restar dias a una fecha.
- **addWeeks(int), subWeeks(int)**: añadir y restar semanas a una fecha.
- **addMonths(int), subMonths(int), addYears(int), subYears(int)**: añadir y restar meses/años a una fecha. Estos métodos cambia el funcionamiento por defecto usado en modify: si estamos en dia 31 y le sumamos un mes, si no existe el dia obtenemos el 1 del mes siguiente. Con estos métodos, en la situación anterior obtenemos el último día del mes siguiente.

También hay métodos para obtener la fecha en distintos formatos o hacer comparaciones:

- **formatUser()**: obtiene la fecha en el formato que ve el usuario. En principio no debería usarse por el programador.
- **formatFW()**: obtiene la fecha en el formato que reconoce el framework. En principio no debería usarse por el programador.
- **formatSOAP()**: obtiene la fecha en el formato usado en SOAP. Lo usaremos para generar fechas en servidores de web services.
- **format($format)**: método de DateTime que acepta los formatos usados por date(). Este método sólo deberia usarse para obtener elementos de la fecha como el dia o el mes. Para otros formatos más complejos habría que valorar si se crea un nuevo método en la clase.

- **isLeap()**: indica si el año de la fecha es bisiesto.
- **cmp($f1, $f2)**: método estático para comparar fechas. Devuelve 1 si la primera en menor, -1 si es mayor y 0 si son iguales. (OBSOLETO, ya que se puede comparar los objetos directamente).
- **between($f1, $f2)**: comprueba si la fecha del objeto se encuentra entre el intérvalo [f1, f2].
- **betweenDays($f1, $days)**: comprueba si la fecha del objeto se encuentra entre el intérvalo [f1, (f1 + days)]. Si el número de dias es negativo se usa el intervalo [(f1 + days), f1]
- **getTimestamp()**: obtiene la fecha en timestamp. Este tipo tiene restricciones (en PHP < 5.3 el rango va de 1970 al 2036 aprox.), por lo que se recomienda no usar (casi todas las operaciones con timestamp se pueden hacer con DateTime).

3.7.3.3. gvHidraDate y gvHidraDatetime

Estas serían las clases usadas para indicar que el tipo de un campo es una fecha, con la diferencia que la segunda admite también la hora. Al definir un campo de este tipo, el framework ya se encarga de hacer las transformaciones necesarias de formatos, asi como de aplicar máscaras y calendarios para la edición.

Estas clases también disponen de los siguientes métodos propios:

- • **setCalendar(bool)**: indica si se muestra o no el calendario.
  - **setDayOfWeek({none|short|long})**: indica si se quiere mostrar el dia de la semana.
  - **setDayOfYear(bool)**: indica si se quiere mostrar el día del año.

Si no se hace uso de los métodos los valores por defecto son *calendario=false, dayOfWeek=none, dayOfYear=false.*

Hay que tener precaución con no usar el tipo gvHidraDate si el campo asociado tiene información de hora en la base de datos, ya que el framework produciria una excepción. Este comportamiento se ha definido de esta forma para evitar pérdidas de datos por truncamiento.

Nota: Hay que tener en cuenta que el tipo gvHidraTimestamp (que hereda de DateTime de PHP) es una marca de tiempo, con lo que vamos a trabajar con instantes de tiempo. En el caso de operaciones con fechas, al crear instancias de dicha clase el valor que coge por defecto es "now" (el instante de tiempo de la invocación); por lo que si se está trabajando con fechas esto puede acarrear errores. Para obtener la referencia al día actual se debe invocar al constructor con el parámetro "today". A continuación mostramos unos ejemplos:

```
// Ejemplo, instante actual 24/07/2009 13:28:30

$f = new gvHidraTimestamp();
print($f->format("d/m/Y H:i:s")); // equivalente a $f->formatUser()
//Resultado = 24/07/2009 13:28:30

$f = new gvHidraTimestamp("today");
print($f->format("d/m/Y H:i:s"));
//Resultado = 24/07/2009 00:00:00

$f = new gvHidraTimestamp("yesterday");
print($f->format("d/m/Y H:i:s"));
//Resultado = 23/07/2009 00:00:00
```

3.7.3.4. Ejemplos

Obtener información de una fecha:

```
$fpeticionIni = $objDatos->getValue('fpeticion');
if (is_null($fpeticionIni))
    return;
$day   = $fpeticionIni->format('d');
$month = $fpeticionIni->format('m');
$year  = $fpeticionIni->format('Y');
```

Comparar fecha:

```
$fpeticionIni = $objDatos->getValue('fpeticion');
if (is_null($fpeticionIni))
   return;
// es fecha futura?
if (new gvHidraTimestamp("today") < $fpeticionIni)
```

Asignar una fecha cualquiera:

```
$f = new gvHidraTimestamp();
$f->setDate(1950,12,24);
$f->setTime(15,0,0);
$objDatos->setValue("fsolucion", $f);
```

Modificar una fecha recibida:

```
$fpeticionIni = $objDatos->getValue('fpeticion');
if (is_null($fpeticionIni))
   return;
$fpeticionIni->subMonths(10);
$objDatos->setValue("fpeticion", $fpeticionIni);
```

Obtener la diferencia en dias entre dos fechas:

```
$diffDias = round(($fini->getTimestamp() - $ffin->getTimestamp()) / (24*60*60)) ;
// a partir de PHP 5.3
$intervalo = $fini->diff($ffin);
$diffDias = $intervalo->format('%a');
```

Usar la fecha en una sentencia sql de la conexión correspondiente:

```
$fechabd = $this->getConnection()->prepararFecha($fpeticionIni);
$sentencia = "update facturas set fecha_entrada = '$fechabd' where usuario = 'xxx'";
```

Obtener un objeto fecha de la base de datos de una consulta que no hace el framework:

```
$this->consultar("SELECT fecha,fechahora from tabla" ,
                 array( 'DATATYPES'=>array('fecha'=>TIPO_FECHA,'fechahora'=>TIPO_FECHAHORA)));
```

3.7.4. Números

3.7.4.1. Entendiendo los números en gvHidra

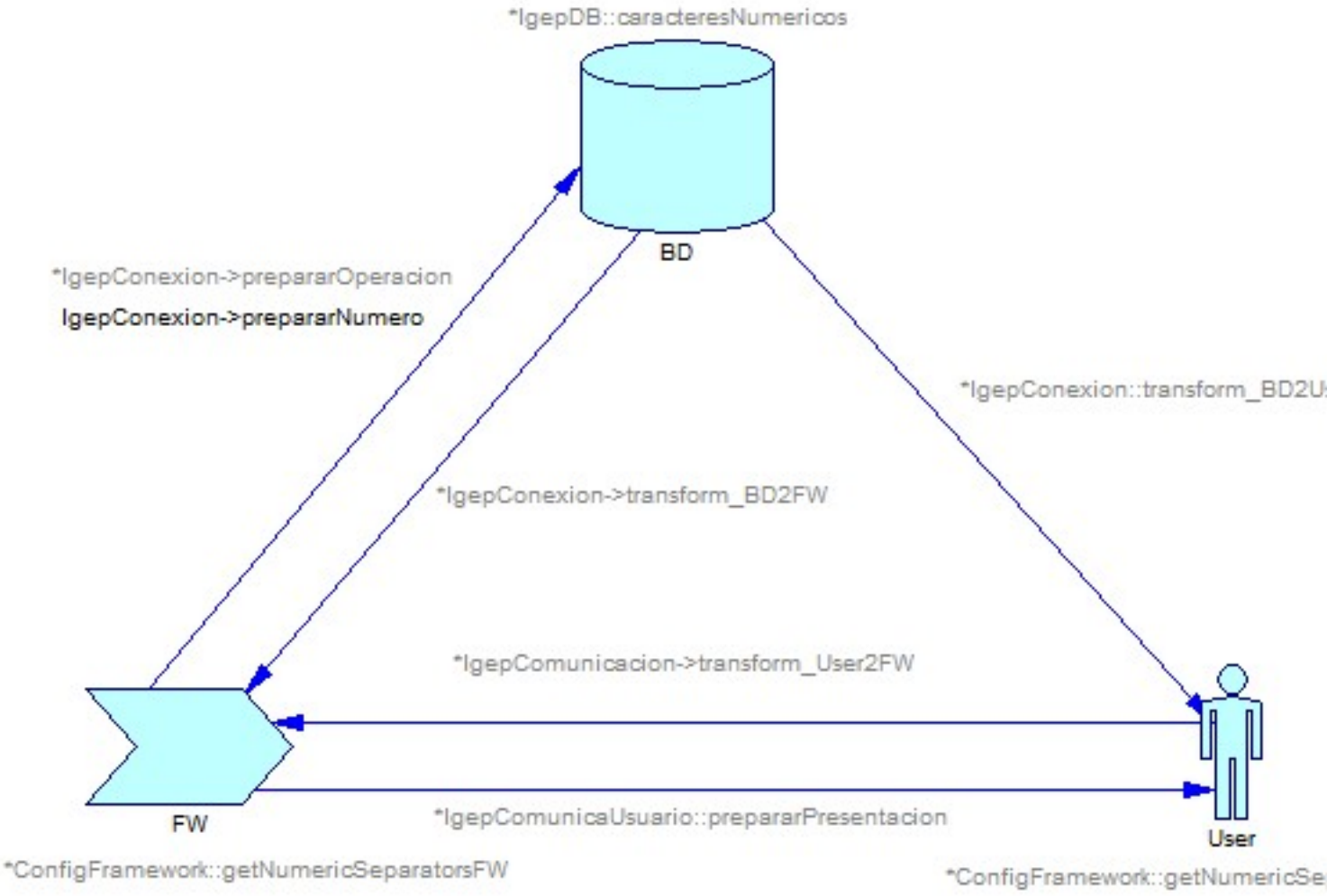
Cuando trabajamos con números podemos encontramos con los mismos problemas que hemos visto para las fechas, y que son los que vienen derivados de los distintos formatos que pueden usarse para representarlos.

En gvHIDRA se definen los formatos numéricos indicando el caracter usado como separador decimal, y el separador de miles (puede ser cadena vacia). Al igual que en las fechas, tenemos:

- formato de interfaz, el usado para interactuar con el usuario, y que viene definido en método **ConfigFramework::getNumericSeparatorsUser**.
- formato de datos, definido para cada tipo de gestor de base de datos, en métodos **caracteresNumericos** de **IgepDBMS_*.** Es el formato usado para las operaciones con las bases de datos de ese tipo.
- formato del framework (o negocio), es el que el programador usa internamente para las operaciones, y se define en **ConfigFramework::getNumericSeparatorsFW**. Este formato coincide con la representación de números interna en PHP, es decir, con punto decimal y sin separador de miles. Por tanto el programador puede manejar los números directamente con los operadores y funciones propios de PHP.

El framework se encarga de hacer las conversiones necesarias en cada operación. De los tres formatos definidos, solo el de la interfaz es configurable por el programador.

En la siguiente figura podemos ver un esquema de la situación, siguiendo las mismas reglas explicadas anteriormente para las fechas:



3.7.4.2. gvHidraInteger y gvHidraFloat

- **gvHidraInteger**.

  Usaremos este tipo para datos de tipo entero. No dispone de métodos propios.

- **gvHidraFloat**.

  Para datos de tipo numérico con coma flotante. Métodos propios:

  - **setFloatLength(int)**. Permite fijar la longitud de la parte decimal del número. De este modo el número se define a partir de su longitud total más la de la parte decimal (siguiendo el patrón de SQL). Si el número de decimales es 0, se recomienda usar gvHidraInteger.

    Nota 1: si no se hace uso del método setFloatLength el valor por defecto es 2.

    Nota 2: Si no coincide el número de decimales en la BD con los que le indicamos al tipo se produce una excepción, por lo que en caso necesario el programador tiene que truncar/redondear los datos según sus necesidades.

3.7.4.3. Uso en el framework

En el constructor de la clase manejadora del panel hay que crear el tipo para poder asignárselo al campo que corresponda, así como marcar las propiedades/características.

```
// numero con 8 digitos en la parte entera y 2 decimales
$tipoNumeroDec =  new gvHidraFloat(false, 10);
// numero con 7 digitos en la parte entera y 3 decimales
$tipoNumeroDec->setFloatLength(3);
$this->addFieldType('ediCoste', $tipoNumeroDec);
```

Primero se crea un objeto de tipo float llamando a la clase gvHidraFloat, con el primer parámetro indicamos la obligatoriedad del campo, y con el segundo el tamaño máximo del campo, incluidos los decimales. Ahora ya podemos hacer uso del método **setFloatLength()**. Y por último se asigna el tipo al campo en concreto, en nuestro caso es "ediCoste" (alias del campo en la select de búsqueda o edición).

De esta forma, gvHidra se encarga de realizar las conversiones necesarias para comunicarse con la BD y el usuario no necesita preocuparse por si debe introducir como separador de decimales la coma o el punto. Los separadores de miles aparecen "solos" y el separador decimal aparece con la coma o el punto del teclado numérico. Cualquier otro caracter (letras, paréntesis, etc...) no pueden introducirse, se ignoran.

Si trabajando en la capa de negocio queremos asignar explícitamente un número para mostrarlo en pantalla, lo podemos hacer directamente con el metodo **setValue()** del objeto datos usando un numero en PHP. Si lo que queremos es coger un número (getValue del objeto datos), operar con él y asignarlo, también lo podemos hacer directamente. Si tras operar con él queremos hacer algo con la base de datos, usaremos el método prepararNumero de la conexión correspondiente:

```
$costebd = $this->getConnection()->prepararNumero($coste);
$sentencia = "update facturas set importe = $costebd where factura = 10";
```

3.7.5. Creación de nuevos tipos de datos

Puede ser interesante que, para ciertos casos puntuales, se requiera crear un tipo de datos concreto que nos garantice que los datos son correctos. El framework realizará las validaciones por nosotros y nos garantizará que el dato es válido. Para ello debemos crear una clase dentro de nuestra aplicación o custom que sea accesible. Esta clase debe heredar de uno de los tipos básicos de gvHidra o del tipo básico (gvHidraTypeBase) si sólo se quieren heredar las características básicas. Esta clase debe implementar la interfaz gvHidraType. Esto supone que el programador de la clase deberá implementar el método validate(). A continuación tenemos un ejemplo:

```
//Type que controla que el campo introducido corresponde con el año actual
class anyoActual extends gvHidraTypeBase implements gvHidraType
{
    public function __construct($required=false)
    {
        $maxLength = 4;
        parent::__construct($required,$maxLength);
    }//Fin de constructor

    public function validate($value)
    {
        if($value!=date('Y'))
           throw new Exception('No ha introducido el año actual.');
    }

}
```

3.8. Listas de datos sencillas

3.8.1. Listas

Las listas de opciones son de gran ayuda para los formularios de las aplicaciones, nos podemos encontrar listas desplegables, listas de tipo radiobutton o listas con checkbox. Vamos a ver como es el uso de estos elementos en gvHidra.

Nos podemos encontrar con listas estáticas o listas dinámicas, básicamente la diferencia se encuentra en el origen de datos que las rellenarán. Su implementación es bastante similar aunque con algunas diferencias. Vamos a describir lo que tienen en común los dos tipos de listas.

Empezaremos viendo lo más sencillo, que es incluir una lista en la plantilla (tpl). El plugin a utilizar es el **CWLista**, vamos a hacer hincapié en el parámetro es "*datos*", el resto se puede ver su definición y uso en el Apéndice Documentación de Plugins. En el parámetro datos nos vendrán los datos que compondrán la lista en cuestión, la variable asignada siempre deberá tener la misma estructura, la que vemos en el ejemplo, siendo la palabra "defaultData_" reservada.

```
{CWLista nombre="codigoProvincia" textoAsociado="Provincia" editable="true" dataType=$dataType_claseManejadora.codigoProvincia
 datos=$defaultData_claseManejadora.codigoProvincia}
```

Pasamos a la parte que le corresponde a la clase manejadora del panel. En ella debemos definir la lista mediante la clase **gvHidraList**, este método define el tipo y contenido de la lista, para ello se le pasan unos parámetros:

1. *Nombre del campo destino de la lista* (campo *obligatorio*). Será el mismo nombre que le hayamos puesto en la tpl al plugin CWLista (p.ej. "codigoProvincia")

2. *Cadena que identifica la fuente de datos* (campo *opcional*). Este campo pasa a ser obligatorio en el caso de las listas dinámicas.

3. *DSN de conexión* (campo *opcional*). Permite incluir un DSN de conexión para indicar una conexión alternativa a la propia del panel. Si no se indica ninguna se cogerá por defecto el DSN del panel. Por ejemplo, podemos tener un panel trabajando en PostgreSQKL y que un campo tenga una lista desplegable que lea los datos en MySQL.

A partir de la clase **gvHidraList()** podemos utilizar los siguientes métodos para acabar de perfilar la lista:

- **addOpcion($valor, $descripcion)**: Nos permite añadir opciones a la lista.

- **setSelected($valor)**: Con este método le indicaremos qué opción aparecerá seleccionada por defecto.

- **setMultiple($multiple)**: Pasándole un parámetro booleano (true/false) indicaremos si es o no una lista múltiple.

- **setSize($size)**: Indicaremos el número de elementos visibles cuando estamos en una lista que es múltiple. Por defecto tendrá un size=5.

- **setDependence($listasCamposTpl, $listasCamposBD,$tipoDependencia=0)**: Método que permite asigar dependencia en una lista, es decir, si tenemos una lista cuyos valores dependen del valor de otros campos, necesitamos indicarlo con este método.

  - *$listasCamposTpl*: Será un array que contiene la lista de campos de la tpl de los cuales depende la lista.Array que, indexado en el mismo orden que el anterior, realiza la correspondencia de los campos del array anterior con los de la Base de Datos.

  - *$listaCamposBD*: Será un array que, indexado en el mismo orden que el anterior, realiza la correspondencia de los campos de la tpl con los de la base de datos.

  - *$tipoDependencia*: Un entero con el que le indicaremos si es una dependencia fuerte->0 o débil->1 (si no tiene valor el campo dependiente lo ignora).

  Nota: No se pueden crear listas dependientes con clausulas group by, ya que gvHidra internamente modifica en cada caso el valor de la where y en estos casos produce un error. De momento se pueden resolver usando una subconsulta con el group by en el from.

```
$listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
$this->addList($listaProvincias);
$listaMunicipios = new gvHidraList('codigoMunicipio','MUNICIPIOS');
$listaMunicipios->setDependence(array('codigoProvincia'),array('tcom_municipios.cpro'));
$this->addList($listaMunicipios);
```

- **setRadio($radio)**: Pasándole un parámetro booleano a true tendremos una lista de radiobuttons.

Una vez hemos definido los datos que compondrán la lista y cómo la queremos, tenemos que añadir la lista al panel, para ello utilizamos el método **addList()** del panel.

```
$listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
$this->addList($listaProvincias);
```

3.8.1.1. Listas estáticas

Como su propio nombre indica, el origen de datos de estas listas vendrá dado por un conjunto de valores indicados de forma estática.

Por lo tanto tendremos que definir la lista únicamente en el constructor de la clase manejadora.

Primero crearemos la lista haciendo uso de la clase **gvHidraList()**, a la que se le pasa por parámetro el nombre del campo que le hemos dado en la tpl. Ahora ya podemos ir añadiendo las opciones que necesitamos, lo haremos con el método **addOption()**, nos permitirá añadir tantas opciones como se necesiten.

A continuación vemos un ejemplo para que sólo aparezcan las provincias pertenecientes a la Comunidad Valenciana:

```
$listaProvincias = new gvHidraList('codigoProvincia');
$listaProvincias->addOption('03','ALICANTE');
$listaProvincias->addOption('12','CASTELLON');
$listaProvincias->addOption('46','VALENCIA');
$listaProvincias->setSelected('12');
$this->addList($listaProvincias);
```

3.8.1.2. Listas dinámicas

En este caso, el conjunto de valores posibles se obtienen a partir de una fuente de datos, ya sea de una consulta a base de datos, una clase...

Tal y como se ha indicado al principio, el segundo parámetro indica la fuente que se va a utilizar para obtener el contenido. Estas fuentes se pueden definir a nivel general de gvHIDRA (consultas o tablas comunes) o a nivel particular de cada aplicación, teniendo en cuenta que prevalecen las definidas en la aplicación sobre las generales del framework, si se han definido con el mismo nombre.

Las definiciones de las listas dinámicas deben ser cargadas al arrancar la aplicación, por ello se deben cargar en el constructor del objeto que maneja el panel principal de la aplicación, es decir, en la clase **AppMainWindow.php**.

Vamos a ver como incluir la fuente de datos, debemos hacer uso del objeto de configuración de gvHidra y del método apropiado según la fuente que carguemos, que pueden ser de dos tipos: fuentes de datos SQL o clases.

- **List_DBSource**

  Esta ha sido la fuente de datos habitual para las listas en gvHIDRA. Básicamente consiste en cargar directamente una query que gvHidra se encargará de parametrizar (añadir filtros a la where) para obtener el resultado esperado.

  En este caso haremos uso del método **setList_DBSource()** donde definiremos la query correspondiente, teniendo en cuenta los siguientes puntos:

  - La consulta ha de seleccionar dos campos, uno se corresponderán con el valor que guardará el campo (le pondremos de alias 'valor' en la query), y el otro corresponderá con la información que visualice la opción de la lista en pantalla (este deberá tener el alias 'descripcion')

  - Por defecto la ordenación es por el campo 'descripcion asc', aunque podemos modificarla con la clausula ORDER BY.

```
class AppMainWindow extends CustomMainWindow
```

```
{
  public function AppMainWindow()
  {
    ...
    $conf = ConfigFramework::getConfig();

    //Tipos
    $conf->setList_DBSource('TIPOS',"select ctipo as \"valor\", dtipo as \"descripcion\" from tinv_tipos");

    //Subtipos
    $conf->setList_DBSource('SUBTIPOS',"select cstipo as \"valor\", dstipo as \"descripcion\" from tinv_subtipos");

    //Estados
    $conf->setList_DBSource('ESTADOS',"select cestado as \"valor\", destado as \"descripcion\" from tinv_estados");
    ...
  }
}
```

En el ejemplo anterior vemos como con el método **setList_DBSource()** añadimos la definición de las fuentes de datos que se gastarán en la aplicación para cargar las listas. Este método tiene dos parámetros, el primero es un identificador de la lista, este identificador debe coincidir con el identificador que se le da a la definición de la lista en la clase manejadora (gvHidraList('nombreCampo','*TIPOS*');); y el segundo es la definición de la consulta SQL tal y como se ha indicado antes, con los alias correspondientes.

- **List_ClassSource**

Estas nos permiten definir como fuente de datos el resultado de un método de una clase. Esta clase, debe implementar la interfaz **gvHidraList_Source** para que el framework pueda interactuar con ella. Consiste en implementar el método *build* que será llamado por el framework para obtener el resultado de la consulta. Aquí introducimos un ejemplo sencillo:

```
/**
 * Fuente de datos ejemplo
 *
 * $Revision: 1.3.2.28 $
 */

class ejemploSource implements gvHidraList_Source
{
    public function __construct()
    {

    }

    public function build($dependence,$dependenceType)
    {

       $resultado = array(
                    array('valor'=>'01','descripcion'=>'UNO'),
                    array('valor'=>'02','descripcion'=>'DOS'),
                    array('valor'=>'03','descripcion'=>'TRES'),
                 );
       return $resultado;
    }
}
```

De este ejemplo cabe destacar:

- La clase implementa la interfaz **gvHidraList_Source**. Si no implementa esta interfaz, no puede ser admitida como fuente de datos de las listas.
- El método **build**, devuelve como resultado un *array*. Este array, puede ser array vacio o un array que contiene por cada una de sus posiciones un array con dos índices válidos (valor y descripción).

Al igual que en el caso de las fuentes de datos tipo List_DBSource(), para incluirlas debemos hacer uso del objeto de configuración de gvHidra llamando al método apropiado.En este caso, el método a utilizar es setList_ClassSource. A continuación mostramos algunos ejemplos de carga.

```
class AppMainWindow extends CustomMainWindow
{
  public function AppMainWindow()
  {
    ...
    $conf = ConfigFramework::getConfig();

    $conf->setList_ClassSource('EJEMPLO','ejemploSource');

    //Subtipos
    $conf->setList_ClassSource('SUBTIPOS',"SubTipos");

    //Estados
    $conf->setList_DBSource('ESTADOS',"EstadosSource");
    ...
  }
}
```

En el ejemplo anterior vemos como con el método **setList_ClassSource()** añadimos la definición de las fuentes de datos que se gastarán en la aplicación para cargar las listas. Este método tiene dos parámetros, el primero es un identificador de la lista, este identificador debe coincidir con el identificador que se le da a la definición de la lista en la clase manejadora (gvHidraList('nombreCampo','*TIPOS*');); y el segundo es el nombre de la clase que actuará como fuente.

Una vez conociendo como funcionan tanto las listas estáticas como las dinámicas, es muy común que se utilice una mezcla de ambas. Este uso nos será útil, por ejemplo, en el siguiente caso:

```
$listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
$listaProvincias->addOption('00','Todas');
$listaProvincias->setSelected('00');
$this->addList($listaProvincias);
```

En el ejemplo añadimos una opción que implica la selección de todas las opciones, hay que tener en cuenta estos valores que se añaden de forma estática, porque al interactuar con la base de datos ese valor no existirá, por lo tanto habrá que añadir un control particular en la clase manejadora.

Otro ejemplo puede ser que el campo de la base de datos admita también nulos, así que tendremos que añadir una opción de valor nulo y descripción en blanco, para darnos la posibilidad de no añadir ningún valor:

```
$listaProvincias->addOption('',' ');
```

3.8.1.3. Listas dependientes

Una vez que conocemos los dos tipos de listas que podemos tener, hay que decir que podemos tener el caso de que el contenido de la lista sea dependiente de otros campos. Para ello, en la definición de las listas hay que indicar cual es la dependencia, lo haremos mediante el método **setDependence()** de la clase **gvHidraList**.

Las definiciones de las listas dinámicas deben ser cargadas al arrancar la aplicación, por ello se deben cargar en el constructor del objeto que maneja el panel principal de la aplicación, es decir, en la clase **AppMainWindow.php**.

Vamos a ver como incluir la fuente de datos, debemos hacer uso del objeto de configuración de gvHidra y del método apropiado según la fuente que carguemos, que pueden ser de dos tipos: fuentes de datos SQL o clases.

- **List_DBSource**

  Esta ha sido la fuente de datos habitual para las listas en gvHIDRA. Básicamente consiste en cargar directamente una query que gvHidra se encargará de parametrizar (añadir filtros a la where) para obtener el resultado esperado.

  En este caso haremos uso del método **setList_DBSource()** donde definiremos la query correspondiente, teniendo en cuenta los siguientes puntos:

  - La consulta ha de seleccionar dos campos, uno se corresponderán con el valor que guardará el campo (le pondremos de alias 'valor' en la query), y el otro corresponderá con la información que visualice la opción de la lista en pantalla (este deberá tener el alias 'descripcion')

  - Por defecto la ordenación es por el campo 'descripcion asc', aunque podemos modificarla con la clausula ORDER BY.

```
class AppMainWindow extends CustomMainWindow
{
  public function AppMainWindow()
  {
    ...
    $conf = ConfigFramework::getConfig();
    //Tipos
    $conf->setList_DBSource('TIPOS',"select ctipo as \"valor\", dtipo as \"descripcion\" from tinv_tipos");

    //Subtipos
    $conf->setList_DBSource('SUBTIPOS',"select cstipo as \"valor\", dstipo as \"descripcion\" from tinv_subtipos");

    //Estados
```

```
    $conf->setList_DBSource('ESTADOS',"select cestado as \"valor\", destado as \"descripcion\" from tinv_estados");
    ...
  }
}
```

En el ejemplo anterior vemos como con el método **setList_DBSource()** añadimos la definición de las fuentes de datos que se gastarán en la aplicación para cargar las listas. Este método tiene dos parámetros, el primero es un identificador de la lista, este identificador debe coincidir con el identificador que se le da a la definición de la lista en la clase manejadora (gvHidraList('nombreCampo','*TIPOS*');); y el segundo es la definición de la consulta SQL tal y como se ha indicado antes, con los alias correspondientes.

- **List_ClassSource**

Estas nos permiten definir como fuente de datos el resultado de un método de una clase. Esta clase, debe implementar la interfaz **gvHidraList_Source** para que el framework pueda interactuar con ella. Consiste en implementar el método *build* que será llamado por el framework para obtener el resultado de la consulta. Aquí introducimos un ejemplo sencillo:

```
/**
 * Fuente de datos ejemplo
 *
 * $Revision: 1.3.2.28 $
 */

class ejemploSource implements gvHidraList_Source
{

    public function __construct()
    {

    }

    public function build($dependence,$dependenceType)
    {

       $resultado = array(
                    array('valor'=>'01','descripcion'=>'UNO'),
                    array('valor'=>'02','descripcion'=>'DOS'),
                    array('valor'=>'03','descripcion'=>'TRES'),
                 );
       return $resultado;
    }//Metodo build
}
```

De este ejemplo cabe destacar:

- La clase implementa la interfaz **gvHidraList_Source**. Si no implementa esta interfaz, no puede ser admitida como fuente de datos de las listas.

- El método **build**, devuelve como resultado un *array*. Este array, puede ser array vacio o un array que contiene por cada una de sus posiciones un array con dos índices válidos (valor y descripción).

Al igual que en el caso de las fuentes de datos tipo List_DBSource(), para incluirlas debemos hacer uso del objeto de configuración de gvHidra llamando al método apropiado.En este caso, el método a utilizar es setList_ClassSource. A continuación mostramos algunos ejemplos de carga.

```
class AppMainWindow extends CustomMainWindow
{
  public function AppMainWindow()
  {
    ...
    $conf = ConfigFramework::getConfig();

    $conf->setList_ClassSource('EJEMPLO','ejemploSource');

    //Subtipos
    $conf->setList_ClassSource('SUBTIPOS',"SubTipos");

    //Estados
    $conf->setList_DBSource('ESTADOS',"EstadosSource");
    ...
  }
}
```

En el ejemplo anterior vemos como con el método **setList_ClassSource()** añadimos la definición de las fuentes de datos que se gastarán en la aplicación para cargar las listas. Este método tiene dos parámetros, el primero es un identificador de la lista, este identificador debe coincidir con el identificador que se le da a la definición de la lista en la clase manejadora (gvHidraList('nombreCampo','*TIPOS*');); y el segundo es el nombre de la clase que actuará como fuente.

Una vez conociendo como funcionan tanto las listas estáticas como las dinámicas, es muy común que se utilice una mezcla de ambas. Este uso nos será útil, por ejemplo, en el siguiente caso:

```
$listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
$listaProvincias->addOption('00','Todas');
$listaProvincias->setSelected('00');
$this->addList($listaProvincias);
```

En el ejemplo añadimos una opción que implica la selección de todas las opciones, hay que tener en cuenta estos valores que se añaden de forma estática, porque al interactuar con la base de datos ese valor no existirá, por lo tanto habrá que añadir un control particular en la clase manejadora.

Otro ejemplo puede ser que el campo de la base de datos admita también nulos, así que tendremos que añadir una opción de valor nulo y descripción en blanco, para darnos la posibilidad de no añadir ningún valor:

```
$listaProvincias->addOption('',' ');
```

3.8.1.4. Uso en acciones

Las listas tienen una serie de métodos especiales que permiten su manipulación en las acciones. Para acceder a una lista desde una acción de este tipo se debe hacer uso de los siguientes métodos:

- **getLista**:obtenemos la lista con la que queremos trabajar.
- **clean**: limpiamos los valores dejándola vacia.
- **setLista**: fijamos una lista.
- **setSelected**: marcamos un valor como seleccionado.
- **toArray**: obtenemos una lista en formato array.
- **arrayToObject**: cargamos un objeto lista con un array.

A continuación mostramos un ejemplo de uso:

```
$lcuentaFe = $objDatos->getLista('cuenta_fe');
$lcuentaFe->clean();

$lcuentaFe->addOpcion('','');

if (count($datos)>0) {

 foreach($datos as $datos_lista) {
     //valor, descripcion
     $lcuentaFe->addOpcion($datos_lista['valor'],$datos_lista['descripcion']);
    }
}
$lcuentaFe->setSelected('');

$objDatos->setLista('cuenta_fe',$lcuentaFe);
...
```

3.8.2. Checkbox

El elemento checkbox podemos emplearlo también para una lista de valores, nos permitirá elegir más de una opción ya que no serán excluyentes. Para definir el checkbox hay que hacer uso de la clase **gvHidraCheckBox** y de sus metodos, que son:

- **setValueChecked($value)**: Fija el valor que se quiere tener cuando el check está marcado.
- **getValueChecked()**: Devuelve el valor que tiene el check cuando está marcado.
- **setValueUnchecked($value)**: Fija el valor que se quiere tener cuando el check está desmarcado.

- **getValueUnchecked()**: Devuelve el valor que tiene el check cuando está desmarcado.
- **setChecked($boolean)**: Indicar si el checkbox aparece marcado o no por defecto.

De este modo en la clase manejadora tendremos que crear el checkbox y definir las propiedades:

```
//Creamos checkbox y lo asociamos a la clase manejadora
$filCoche = new gvHidraCheckBox('filCoche');
$filCoche->setValueChecked('t');
$filCoche->setValueUnchecked('f');
$this->addCheckBox($filCoche);
```

En la tpl definiremos el checkbox de la siguiente forma:

```
{CWCheckBox nombre="filCoche" editable="true" textoAsociado="Coche" dataType=$dataType_claseManejadora.filCoche
valor=$defaultData_claseManejadora.filCoche}
```

Es importante añadir el parámetro dataType que es el que asociará la definición dada en la clase manejadora con el plugin.

3.9. Mensajes y Errores

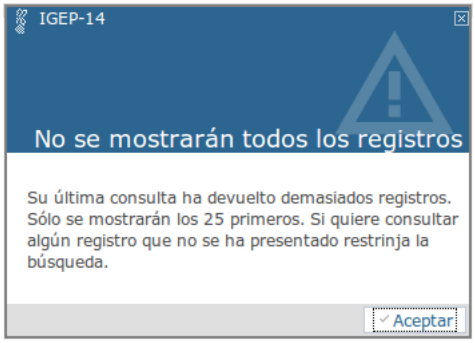
En cualquier aplicación es necesario mostrar avisos, errores... En gvHidra todo este tipo de mensajes se ha clasificado en cuatro grupos, más que nada por su aspecto visual, ya que todas serán invocadas de la misma forma, asociando un color a un tipo de mensaje:

1. **Alertas**.

   Este tipo de mensajes será útil para alertar al usuario de un estado, aunque pueda continuar trabajando pero con conocimiento de un estado. En el ejemplo se alerta de que la aplicación está en desarrollo por lo tanto puede ser inestable.

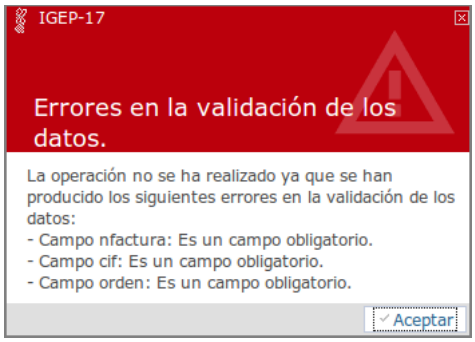
2. **Avisos**.

   Este tipo de mensaje nos avisa de cierta situación, nada problemática, pero que tengamos en cuenta.



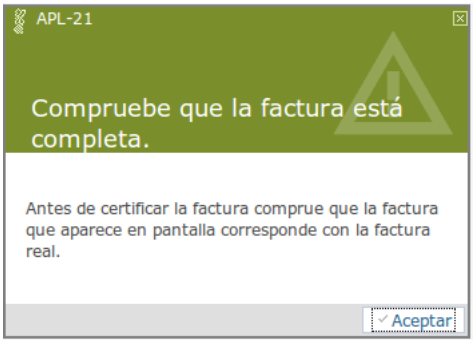
3. **Errores**.

Claramente los errores muestran problemas ocurridos al efectuar una acción. En el ejemplo se avisa de que hay que rellenar ciertos campos de forma obligatoria y por eso falla la búsqueda.



4. **Sugerencias**.

   Son mensajes de tipo consejo al usuario que no le impiden continuar con el trabajo. En el ejemplo se aconseja al usuario cerciorarse de que todo esté rellenado antes de efectuar la acción.



Hay *mensajes que son propios del framework*, son mensajes generales a cualquier aplicación, se encuentran ubicados en la clase **IgepMensaje.php**. Por otro lado estarán los *mensajes particulares de cada aplicación*, estos se definirán en el fichero **mensajes.php**, este fichero se encuentra en el directorio raíz de la aplicación.

Vamos a explicar como añadir mensajes en el fichero *mensajes.php*. En este fichero existe una variable global que es el array donde se irán almacenando los mensajes (**$g_mensajesParticulares**).

```
<?php

global $g_mensajesParticulares;
$g_mensajesParticulares = array(
     'APL-1'=>array('descCorta'=>'No se puede realizar el borrado','descLarga'=>'No se puede borrar un tipo que tiene subtipos asociados. Si quiere eliminar este tipo deberá borrar
 todos sus subtipos.','tipo'=>'ERROR'),
     'APL-2'=>array('descCorta'=>'Se ha listado la factura.','descLarga'=>'Se ha listado la factura %0%-%1%.','tipo'=>'AVISO'),
...
);

?>
```

Un mensaje se crea añadiendo un elemento al array asociativo. El elemento tiene una clave única (ej. 'APL-1'), ya que esta clave es la que se utilizará como identificador para invocarlo, y como valor es otro array que contiene tres elementos:

1. **descCorta**: Descripción corta del mensaje, esta descripción aparecerá en la parte superior del mensaje, en la zona coloreada.

2. **descLarga**: Descripción completa del mensaje, esta descripción aparecerá en la parte inferior del mensaje, zona blanca. Aquí podemos jugar con el texto del mensaje y pasarle parámetros desde su invocación, nos dará un mensaje más personalizado. Los valores vendrán en un array, y aquí se hará referencia a ellos de la siguiente forma %0% para el primer valor del array, %1% para el segundo, y así sucesivamente.

3. **tipo**: palabra clave que definirá el tipo del mensaje. Estas palabras pueden ser: AVISO, ERROR, SUGERENCIA y ALERTA, que se corresponden con los cuatro grupos vistos anteriormente.

La invocación de los mensajes se puede efectuar desde dos puntos distintos, desde código o mediante parámetro del plugin.

3.9.1. Invocación desde código

Es el uso más habitual. Para invocar un mensaje hay que hacer uso del método **showMensaje()** y pasándole como parámetro el identificador del mensaje que queremos mostrar, que debe coincidir con el definido en *mensajes.php*:

```
if ( <condicion> ) {
   $this->showMensaje('APL-3');
   return 0;
}
```

También podemos pasar argumentos a los mensajes, atributos para personalizar mejor el mensaje que queremos dar. En este caso, en el texto del mensaje deberemos colocar las variables a sustituir, de la forma '%0%', '%1%', ... tal y como hemos explicado anteriormente. Estas variables se pasan en el método **showMensaje()** mediante un array con tantos valores como variables hayamos definido. A continuación se muestra un ejemplo:

```
// APL-4: El valor ha de ser mayor de %0% y menor de %1%
if ( <condicion> ) {
    $this->showMensaje('APL-4', array('5','10'));
    return 0;
}
```

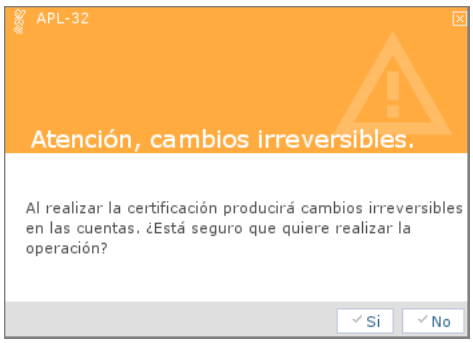
3.9.2. Invocación como confirmación

En este caso nos referimos a ventanas que solicitan del usuario una confirmación para continuar con la acción o cancelarla.

Para crear este tipo de mensajes de confirmación hay que añadir el parámetro "*confirm*" al plugin **CWBoton** (ver Apéndice X Documentación plugins) indicando el identificador del mensaje que queremos mostrar (ej. "APL-2").

De este modo, nos aparecerá un mensaje con dos alternativas Si/No. Al pulsar Si se ejecutará la acción, al pulsar No, la acción quedará cancelada.

```
{CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar" confirm="APL-32"}
```



Capítulo 4. Elementos de pantalla avanzados

4.1. Patrones complejos

4.1.1. Maestro/Detalle

4.1.1.1. Maestro/Detalle

Un maestro-detalle es el caso de una ventana con paneles dependientes, este caso no se diferencia mucho de una ventana normal de gvHidra aunque tiene sus peculiaridades.

Al igual que en las ventanas "normales", en primer lugar se debe crear una clase por cada uno de los paneles que aparezcan en la pantalla, es decir, una clase para el maestro y una para el detalle. Como siempre estas clases se depositarán en el directorio actions de la estructura de la aplicación.

Las clases, tanto la del maestro como la del detalle, son como cualquier clase de un panel, tal y como se ha explicado en XXX, salvo unas anotaciones:

- *Clase manejadora del maestro*.

  Un panel que tenga otro/s panel/es dependiente/s tendrán que utilizar el método **addHijo()** en el constructor de la clase. A este método se le pasan tres parámetros, el primero es el nombre de la clase hija, el segundo y tercer parámetro son dos arrays que definirán por qué campos se establece la relación entre maestro y detalle. El segundo es un array con los campos del padre que van a ser utilizados para recargar el hijo, y el tercer parámetro, un array con el nombre de los campos del detalle que se corresponden con los del maestro.

```
class TinvTipos extends gvHidraForm_DB
{
   function TinvTipos()
   {
      ...
      $this->addHijo("TinvSubtipos",array("lisCodigoTipo"),array("ediCodigoTipo"));
      ...
   }
}
```

  Por ejemplo, tenemos que crear un mantenimiento para dos tablas: tipos y subtipos. Estas tablas tienen una relación 1:N (por cada tipo pueden haber N subtipos). Implementado en gvHidra, tendremos dos clases en el directorio actions (p.e. TinvTipos y TinvSubtipos). La clase *TinvTipos* tiene un hijo que se llama *TinvSubtipos*. El campo de la tpl *ediCodigoTipo* (en el panel maestro), se utiliza para identificar a los subtipos (los detalles) y en el panel detalle existe un campo que hace referencia al campo del maestro y cuyo nombre es lisCodigoTipo.

- *Clase manejadora del detalle*.

  Igual que en el maestro, el detalle también tiene que tener una referencia a su padre. Concretamente se debe utilizar el método **addPadre()** en el constructor de la clase, donde se le pasará como parámetro el nombre de la clase que maneja el panel maestro. Por tanto, siguiendo con nuestro ejemplo, la clase TinvSubtipos debe incorporar la siguiente línea:

```
class TinvSubtipos extends gvHidraForm_DB
{

     function TinvSubtipos()
     {
        ...
        $this->addPadre("TinvTipos");
        ...
     }
}
```

El siguiente cambio respecto a las ventanas "normales" viene en la creación de los paneles en los ficheros ubicados en el directorio *views*. Deberemos definir los dos paneles, maestro y detalle, a partir de la clase IgepPanel. Una vez definidos

debemos utilizar el método de la clase **IgepPantalla, agregarPanelDependiente()**, para agregar el panel detalle al maestro. A este método se le han de pasar dos parámetros, el primero debe ser el panel a agregar y, el segundo, el nombre de la clase que maneja el panel padre. El fichero de views podría quedar así:

```
<?php
$comportamientoVentana = new IgepPantalla();

// Definición del MAESTRO
$panelMaestro = new IgepPanel('TinvTipos',"smty_datosTablaM");
$panelMaestro->activarModo("fil","estado_fil");
$panelMaestro->activarModo("lis","estado_lis");
$comportamientoVentana->agregarPanel($panelMaestro);

// Definición del DETALLE
$panelDetalle = new IgepPanel('TinvSubtipos',"smty_datosFichaD");
$panelDetalle->activarModo("edi","estado_edi");
// Agregamos el panel detalle al maestro
$comportamientoVentana->agregarPanelDependiente($panelDetalle,"TinvTipos");

$s->display('tablasMaestras/p_tiposubtipo.tpl');
?>
```

Finalmente, en la plantilla (tpl) para un maestro detalle también tiene algunas características que destacar de una plantilla para una ventana sencilla. Vamos a ver los puntos:

- Hay que indicar que es un panel maestro con el atributo **esMaestro** del plugin **CWPanel** (esMaestro="true")
- El parámetro **itemSeleccionado** del plugin **CWPanel** es un parámetro interno para el correcto funcionamiento de gvHidra, por eso se asigna a una variable smarty, **itemSeleccionado=$smty_filaSeleccionada**, para mantener el maestro seleccionado.
- El botón tooltip (**CWBotonTooltip**) para insertar registros en el maestro tiene un parámetro **ocultarMD** con el que le indicamos qué panel se debe ocultar cuando pasamos a modo inserción, ya que va a ser un registro nuevo del maestro por lo tanto aún no tiene detalle.
- En el plugin **CWContenedorPestanyas** se tiene que indicar con el parámetro **id** si las pestañas son del maestro o del detalle.
- Las pestañas del panel maestro (**CWPestanya**) tienen unos parámetros, **mostrar** y **ocultar**, con los que se indicará si el panel detalle lo queremos visible o no. Además está el parámetro **panelAsociado** donde se indica a qué panel se refiere esa pestaña, ya que vamos a tener pestañas tanto para el maestro como para el detalle.
- Llegamos a la parte de definir el detalle, establecemos una condición con sintaxis de smarty **{if count($smty_datosTablaM) gt 0}**, para que no nos muestre el panel detalle si no hay datos en el maestro.
- El identificador, **id**, que le demos al panel del detalle (**CWPanel**) deberá ser lisDetalle, en el caso de estar en un modo tabular, o ediDetalle, en modo registro. También hay que indicar el parámetro **detalleDe** pasándole el **id** del maestro.
- Tanto si el detalle es una ficha (**CWFichaEdicion**) como un tabular (**CWTabla**) hay que definir su **id**, *FichaDetalle* o *TablaDetalle*, respectivamente.
- En el panel del detalle es necesario definir el o los campos que contengan la clave del maestro. En el siguiente ejemplo tenemos un maestro-detalle, siendo la clave del maestro el campo "*lisCodigoTipo*", el detalle necesita del valor de este campo, para ello vemos que en el panel del detalle tenemos un campo "*ediCodigoTipo*", que será el que nos relacione el detalle con su maestro. Para que el campo del detalle tenga el valor de la clave del maestro hay que definir su propiedad value, en el ejemplo **value=$defaultData_TinvSubtipos.ediCodigoTipo**.

  Si nos encontramos en el caso de un detalle "Tabular-Registro" hay que definir estos campos clave tanto para el panel tabular como para el panel registro.

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
{CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=$smty_customTitle}
  {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
{/CWBarra}

{CWMarcoPanel conPestanyas="true"}

<!-- ********************** PANEL MAESTRO **********************-->
<!--*********** PANEL fil ******************-->
  {CWPanel id="fil" action="buscar" method="post" estado=$estado_fil  claseManejadora="TinvTipos"}
```

```
    {CWBarraSupPanel titulo="Tipos de Bienes"}
       {CWBotonTooltip imagen="04" titulo="Limpiar Campos" funcion="limpiar" actuaSobre="ficha"}
    {/CWBarraSupPanel}
    {CWContenedor}
       {CWFicha}
            <br />
            {CWCampoTexto nombre="filCodigoTipo" editable="true" size="2" textoAsociado="Código de Tipo" dataType=$dataType_TinvTipos.filCodigoTipo}
          <br /><br />
            {CWCampoTexto nombre="filDescTipo" editable="true" size="30" textoAsociado="Descripción de Tipo" dataType=$dataType_TinvTipos.filDescTipo}
          <br /><br />
       {/CWFicha}
    {/CWContenedor}
    {CWBarraInfPanel}
     {CWBoton imagen="50" texto="Buscar" class="boton" accion="buscar"}
    {/CWBarraInfPanel}
  {/CWPanel}

<!--********** PANEL lis ******************-->
  {CWPanel id="lis" tipoComprobacion="envio" esMaestro="true" itemSeleccionado=$smty_filaSeleccionada action="operarBD" method="post" estado=$estado_lis
 claseManejadora="TinvTipos"}
    {CWBarraSupPanel titulo="Tipos de Bienes"}
       {CWBotonTooltip imagen="01" titulo="Insertar registros" funcion="insertar" actuaSobre="tabla" ocultarMD="Detalle"}
       {CWBotonTooltip imagen="02" titulo="Modificar registros" funcion="modificar" actuaSobre="tabla"}
       {CWBotonTooltip imagen="03" titulo="Eliminar registros" funcion="eliminar" actuaSobre="tabla"}
         {CWBotonTooltip imagen="04" titulo="Limpiar Campos" funcion="limpiar" actuaSobre="tabla"}
    {/CWBarraSupPanel}
    {CWContenedor}
     {CWTabla conCheck="true" conCheckTodos="false" id="Tabla1" numFilasPantalla="4" datos=$smty_datosTablaM}
        {CWFila tipoListado="false"}
           {CWCampoTexto nombre="lisCodigoTipo" editable="nuevo" size="2" textoAsociado="Cod.Tipo" dataType=$dataType_TinvTipos.lisCodigoTipo}
           {CWCampoTexto nombre="lisDescTipo" editable="true" size="25" textoAsociado="Tipo" dataType=$dataType_TinvTipos.lisDescTipo}
        {/CWFila}
        {CWPaginador enlacesVisibles="3"}
     {/CWTabla}
    {/CWContenedor}
    {CWBarraInfPanel}
     {CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
     {CWBoton imagen="42" texto="Cancelar" class="boton" accion="cancelar"}
    {/CWBarraInfPanel}
  {/CWPanel}

<!-- ****************** PESTAÑAS ************************-->
  {CWContenedorPestanyas id="Maestro"}
     {CWPestanya tipo="fil" panelAsociado="fil" estado=$estado_fil ocultar="Detalle"}
     {CWPestanya tipo="lis" panelAsociado="lis" estado=$estado_lis mostrar="Detalle"}
  {/CWContenedorPestanyas}
</td></tr>
<tr><td>
<!-- ****************** PANEL DETALLE ************************-->
  {if count($smty_datosTablaM) gt 0}
   {CWPanel id="ediDetalle" tipoComprobacion="envio" action="operarBD" detalleDe="lis" method="post" estado="on" claseManejadora="TinvSubtipos"}
   {CWBarraSupPanel titulo="Subtipos de Bienes"}
      {CWBotonTooltip imagen="01" titulo="Insertar registros" funcion="insertar" actuaSobre="ficha"}
      {CWBotonTooltip imagen="02" titulo="Modificar registros" funcion="modificar" actuaSobre="ficha"}
      {CWBotonTooltip imagen="03" titulo="Eliminar registros" funcion="eliminar" actuaSobre="ficha"}
   {/CWBarraSupPanel}
   {CWContenedor}
      {CWFichaEdicion id="FichaDetalle" datos=$smty_datosFichaD}
         {CWFicha}
            <br /><br />
            {CWCampoTexto nombre="ediCodigoSubtipo" editable="nuevo" size="3" textoAsociado="Código" dataType=$dataType_TinvSubtipos.ediCodigoSubtipo}
            <br /><br />
            {CWCampoTexto nombre="ediDescSubtipo" editable="true" size="35" textoAsociado="Descripción" dataType=$dataType_TinvSubtipos.ediDescSubtipo}
            <br /><br />
            {CWCampoTexto nombre="ediCodigoTipo" oculto="true" value=$defaultData_TinvSubtipos.ediCodigoTipo}
         {/CWFicha}
         {CWPaginador enlacesVisibles="3"}
      {/CWTabla}
   {/CWContenedor}
   {CWBarraInfPanel}
      {CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
      {CWBoton imagen="42" texto="Cancelar" class="boton" accion="cancelar"}
   {/CWBarraInfPanel}
  {/CWPanel}
<!-- ****************** PESTAÑAS ************************-->
  {CWContenedorPestanyas id="Detalle"}
   {CWPestanya tipo="lis" panelAsociado="ediDetalle" estado=$estado_edi}
  {/CWContenedorPestanyas}
 {/if}
{/CWMarcoPanel}
{/CWVentana}
```

Una vez tenemos claro como definir los ficheros correspondientes a la pantalla, vamos a comentar un aspecto del **mappings.php** respecto a un maestro/detalle. Respecto al maestro, hay que añadir una entrada que permita recargar el detalle a partir del registro seleccionado en el maestro, para ello se deben agregar las siguientes líneas en el mappings.php para la clase del maestro:

```
$this->_AddMapping('TinvTipos__recargar', 'TinvTipos');
$this->_AddForward('TinvTipos__recargar', 'gvHidraSuccess', 'index.php?view=views/patronesMD/M(FIL-LIS)-D(LIS)/p_tiposubtipo.php&panel=listar');
$this->_AddForward('TinvTipos__recargar', 'gvHidraError', 'index.php?view=views/patronesMD/M(FIL-LIS)-D(LIS)/p_tiposubtipo.php&panel=listar');
```

4.1.1.2. Maestro/NDetalles

En este caso nos encontramos con un maestro con más de un detalle, visualmente tendremos un panel superior que corresponderá con el maestro, y en la parte inferior tendremos los detalles, estos serán accesibles mediante unas solapas superiores (ver imagen en el capítulo X Características técnicas).

En general, la forma de funcionar es muy parecida al maestro-detalle explicado en el punto anterior, salvo pequeñas diferencias. Igual que siempre, tenemos que definir una clase por cada panel, es decir, una por el maestro y otra por cada detalle que vayamos a tener. En este caso, en el maestro tenemos que añadir la referencia a todos los detalles que se vayan a tener con el método **addHijo()**.

```
class Expedientes extends gvHidraForm_DB
{
   function Expedientes()
   {
      ...
      $this->addHijo('informes',array('ediNexpediente'),array('lisNexpediente'));
      $this->addHijo('deficiencias',array('ediNexpediente'),array('lisNexpediente'));
      ...
   }
}
```

Los paneles detalle añadirán la referencia al padre igual que en el caso de un maestro-detalle simple, con el método **addPadre()**.

En el fichero de views tendremos que definir todos los paneles detalle que tengamos y agregarlos con el método **agregarPanelDependiente()**. Otra peculiaridad de este panel es que tenemos que definir las solapas que irán asociadas a los paneles detalle, para ello tenemos que definir un array asociativo con tantos arrays como detalles, estos deben tener dos claves, "*panelActivo*" y "*titDetalle*", que se corresponden, respectivamente, con el nombre de la clase manejadora del detalle y el texto que queramos que aparezca en la solapa. Otro punto a tener en cuenta es indicar qué detalle queremos que aparezca activo, para ello tenemos que asignar a la variable smarty *smty_panelActivo* el nombre de la clase manejadora que queramos.

```
<?php
//MAESTRO
$comportamientoVentana= new IgepPantalla();
$panelMaestro = new IgepPanel('expedientes',"smty_datosTablaM");
$panelMaestro->activarModo("fil","estado_fil");
$panelMaestro->activarModo("edi","estado_edi");
$datosPanel = $comportamientoVentana->agregarPanel($panelMaestro);

//DETALLE
$panelDetalle = new IgepPanel('informes',"smty_datosInformes");
$panelDetalle->activarModo("lis","estado_lis");
$datosPanelDetalle = $comportamientoVentana->agregarPanelDependiente($panelDetalle,"expedientes");

$panelDetalle = new IgepPanel('deficiencias',"smty_datosDeficiencias");
$panelDetalle->activarModo("edi","estado_edi");
$datosPanelDetalle = $comportamientoVentana->agregarPanelDependiente($panelDetalle,"expedientes");

// Botones detalles
// panelActivo: Nombre de la clase manejadora del detalle q vamos a activar
// titDetalle: Título q aparecerá en la pestaña del panel
$detalles = array (
      array (
            "panelActivo" =>"informes",
            "titDetalle" =>"Informes"
      ),
      array (
            "panelActivo" =>"deficiencias",
            "titDetalle" =>"Deficiencias"
      )
);
$s->assign('smty_detalles',$detalles);

$s->assign('smty_panelActivo','informes');

$s->display('MaestroNDetalles/p_expedientes.tpl');
?>
```

En la tpl también hay que señalar algunas diferencias respecto a un maestro-detalle simple. La parte que corresponde al maestro es exactamente igual que la explicada para el maestro-detalle, las diferencias las encontramos en la parte que corresponde a los detalles. Vamos a verlo por puntos:

- Vemos en la sección detalles se comprueba, mediante un if de smarty, si en el maestro hay datos. Si los hay pasamos a dibujar las solapas de los diferentes detalles con el plugin **CWDetalles**.
- A continuación englobamos cada panel detalle dentro de un if condicional que comprueba si el panel es el activo por defecto.

```
 1 {CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
   {CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=$smty_customTitle}
   {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
   {/CWBarra}
```

```
 5
   {CWMarcoPanel conPestanyas="true"}

   <!-- ********************** PANEL MAESTRO ***********************-->
   <!--********** PANEL fil ******************-->
10  {CWPanel id="fil" action="buscar" method="post" estado=$estado_fil  claseManejadora="expedientes"}
    ...
    {/CWPanel}

   <!-- ***************** PANEL edi ***********************-->
15  {CWPanel id="edi" tipoComprobacion="envio" esMaestro="true" itemSeleccionado=$smty_filaSeleccionada action="operarBD" method="post" estado="$estado_edi"
claseManejadora="expedientes" accion=$smty_operacionFichaexpedientes}
    ...
    {/CWPanel}

   <!-- ***************** PESTAÑAS ***********************-->
20  {CWContenedorPestanyas id="Maestro"}
      {CWPestanya tipo="fil" panelAsociado="fil" estado=$estado_fil ocultar="Detalle"}
      {CWPestanya tipo="edi" panelAsociado="edi" estado=$estado_edi mostrar="Detalle"}
    {/CWContenedorPestanyas}

25 <!-- ***************** PANELES DETALLES ***********************-->
   </td></tr>
   {if count($smty_datosTablaM ) gt 0 }
    {CWDetalles claseManejadoraPadre="expedientes" detalles=$smty_detalles panelActivo=$smty_panelActivo}
   <tr><td>
30
   <!-- ***************** Detalle 1: INFORMES ***********************-->
    {if $smty_panelActivo eq "informes"}
      {CWPanel id="lisDetalle" tipoComprobacion="envio" action="operarBD" method="post" detalleDe="edi" estado="on" claseManejadora="informes"}
      ...
35    {/CWPanel}
      {CWContenedorPestanyas  id="Detalle"}
         {CWPestanya tipo="lis" panelAsociado="lisDetalle" estado="on"}
      {/CWContenedorPestanyas}
    {/if}
40
   <!-- ***************** Detalle 2: DEFICIENCIAS ***********************-->
    {if $smty_panelActivo eq "deficiencias" }
      {CWPanel id="ediDetalle" tipoComprobacion="envio" action="operarBD" method="post" detalleDe="edi" estado="on" claseManejadora="deficiencias"}
      ...
45    {/CWPanel}
      {CWContenedorPestanyas  id="Detalle"}
         {CWPestanya tipo="edi" panelAsociado="ediDetalle" estado="on"}
      {/CWContenedorPestanyas}
    {/if}
50
   {/if}
   {/CWMarcoPanel}
   {/CWVentana}
```

4.1.2. Árbol

Vamos a explicar ahora como desarrollar un mantenimiento tipo árbol, uno de los patrones más complejos. El aspecto visual del árbol es el siguiente:

La pantalla se divide claramente en dos zonas. La zona de la derecha se corresponde con lo que es el árbol en sí, y la de la izquierda corresponde con un panel común (tabular o registro). El usuario puede ir navegando por el árbol, expandiendo los nodos tal y como se haya especificado en el código, una vez llegado a la opción que nos interese en la parte derecha aparecerá un panel que visualizará los datos correspondientes al nodo seleccionado (p.e. en la imagen se puede ver que se muestra una factura del tercer trimestre para el año 1993, que es el nodo seleccionado).

4.1.2.1. ¿Cómo definir la estructura del árbol?

Primero hay que tener claro como va a ser la estructura del árbol, para poder definir los nodos y hojas del árbol. Estás definiciones, básicamente, se hacen declarando los "tipos" de nodos y como se obtienen sus hijos, indicando qué nodos aparecerán en la raiz, y los que vayan a ser hojas no tendrán especificación de sus hijos. De esta forma obtenemos un árbol muy flexible, en el que no hay que preocuparse de los niveles que va a tener.

Para definir un nodo (o nodo raiz) tenemos que especificar como se obtienen sus hijos, esto se puede hacer de dos modos:

- **SELECT**: Los hijos se obtienen a través de una consulta a base de datos.
- **LISTA**: Los hijos son fijos, se establecen mediante un array de valores.

Con estos dos modos de definición de nodos podemos ir dando la estructura que queramos al árbol.

Como en el resto de pantallas, el programador tendrá que completar un fichero TPL, un fichero PHP de views y, al menos dos, ficheros PHP de actions. Vamos a ir explicando la estructura y como trabajar en cada uno de estos ficheros.

4.1.2.1.1. Fichero de Actions

Lo primero que se debe crear es un fichero PHP donde se va a definir toda la estructura del árbol. Este fichero extenderá de la clase **gvHidraTreePatern**, esta clase nos proporcionará los métodos adecuados para crear el árbol.

El comienzo de la clase es igual que cuando trabajamos con cualquier otro tipo de panel, cargando la conexión, tablas con las que trabajar... Pasamos a crear el árbol en sí, para ello creamos un objeto de la clase **IgepArbol** a partir del que definiremos la estructura con los métodos:

- **addNodoRaiz($tipoNodo, $etiqueta, $tipoDespliegue, $datosDespliegue, $conexion)**

  Con este método crearemos el nodo raíz, hay que tener en cuenta que podemos tener más de un nodo raíz. Los parámetros que se necesitan para definirlo son:

  - *$tipoNodo*: Nombre con el que se identificará el nodo.
  - *$etiqueta*: Texto que aparecerá en pantalla este nodo. Importante no introducir retornos de carro.
  - *$tipoDespliegue*: Nos indica el modo de obtención de los nodos hijos, puede tomar dos valores "SELECT", nodos hijos vienen de BD, o "LISTA", nodos hijos fijados por un array.
  - *$datosDespliegue*: Dependiendo de lo dicho en $tipoDespliegue.
    - Si tenemos "*SELECT*": nos encontraremos un array en la que el primer parámetro se corresponderá con el identificador del primer nivel del árbol, y el segundo parámetro será la select que obtendrá los valores correspondientes a este primer nivel. En la select tenemos que definir el texto que aparecerá en el árbol y darle el alias "etiqueta".

```
$arbol->addNodoRaiz("ANYOS",
                    "A&ntilde;os de Facturas",
                    "SELECT",
                    array(
                         "ANYO",
                         "select anyo||' ('||count (1)||')' as \"etiqueta\",anyo from tinv_entradas group by anyo")
                    );
```

    - Si tenemos "*LISTA*": Tendremos un array asociativo en el que se irá dejando como índice la etiqueta de los nodos y como valor el tipo de hijo correspondiente.

```
$arbol->addNodoRaiz('DECADAS',
                    'A&ntilde;os',
                    'LISTA',
                    array(
```

```
                'D&eacute;cada de los 70'=>'1970',
                'D&eacute;cada de los 80'=>'1980',
                'D&eacute;cada de los 90'=>'1990')
            );
```

- *$conexion*: Se puede indicar una conexión alternativa a la propia del panel para el despliegue de dicha raíz.

- **addNodoRama($tipo,$modoDespliegue,$despliegue,$dsnAlternativo = '')**

  La diferencia entre éste y el anterior es que no tiene el parámetro *$etiqueta*, ya que la etiqueta sale de *$datosDespliegue* del padre. Todos los nodos del árbol excepto los que estén en la raíz se definen con este método.

- **setNodoPanel($tipoNodo, $claseManejadora, $dependencia, $conexion)**

  Este método permite al programador indicar que cierto tipo de nodo tiene asociada una representación en un panel asociado al arbol. Los parámetros que se necesitan en este método son:

  - *$tipoNodo*: Indicamos el tipo de nodo que se va a representar, que debe coincidir con alguno de los definidos con el método *addNodoRama()*.

  - *$claseManejadora*: Nombre de la clase manejadora que define ese panel.

  - *$dependencia*: Un array con los campos que debe coger de los nodos superiores para pasárselos al panel antes de realizar la búsqueda.

  - *$conexion*: Posibilidad de establecer una conexión alternativa para este panel en concreto.

A continuación vemos un ejemplo de como se crea la estructura de un árbol:

```
<?php

class PruebaArbol extends gvHidraTreePattern
{

  function PruebaArbol()
  {
      $conf = ConfigFramework::getConfig();
      $g_dsn = $conf->getDSN('g_dsn');
      $nombreTablas= array("tinv_entradas");
      parent::__construct($g_dsn,$nombreTablas);

      $arbol = new IgepArbol();
      $arbol->addNodoRaiz("ANYOS","A&ntilde;os de Facturas","SELECT",array("ANYO","select anyo||' ('||count (1)||')' as \"etiqueta\",anyo from tinv_entradas group by anyo"));

      $arbol->addNodoRama("ANYO","LISTA",array("Por DG" =>"DGS","Por Trimestre"=>"TRIMESTRES","Ver Todas"=>"TODAS"));
      $arbol->addNodoRama("DGS","SELECT",array("FACTURA-DG","SELECT anyo,cdg,cdg as \"etiqueta\" FROM TINV_ENTRADAS WHERE anyo = '%anyo%' group by anyo,cdg order by
 cdg",array("anyo")));
      $arbol->addNodoRama("FACTURA-DG","SELECT",array("FACTURA","SELECT anyo||'-'||nfactura as \"etiqueta\",anyo,nfactura FROM TINV_ENTRADAS WHERE anyo = '%anyo%' and cdg ='%cdg
%'",array("anyo","cdg")));
      ...
      $arbol->setNodoPanel("ANYO","TinvFacturasArbol",array("anyo"),"Facturas de %anyo%");
      $arbol->setNodoPanel("FACTURA-DG","TinvFacturasArbol",array("anyo","cdg"),"Facturas de %anyo% de la %cdg%");
      ...
      $this->addArbol($arbol);
      ...
}
?>
```

Además de este fichero que crea la estructura principal, se debe crear un fichero php en el directorio actions por cada nodo panel que hayamos definido. Estas clases ya son clases manejadoras de un panel convencional (p.ej. un tabular, registro...).

4.1.2.1.2. Fichero de Views

A continuación debemos crear un archivo php en el directorio **views** que controle la presentación. En él se definen las asignaciones de los datos a la plantilla (tpl). En este caso no es un comportamiento genérico por lo tanto hay que instanciar la clase **IgepPanelArbol** (línea 3) que define el comportamiento del árbol, al que se le pasan los mismos parámetros que una pantalla genérica, el primero será el nombre de la *clase manejadora* que corresponde a ese panel, y el segundo, *smty_datosPanel*, variable smarty que se habrá definido en la tpl correspondiente al árbol. Activaremos la pestaña, método **activarModo**. Finalizamos agregando el árbol al panel con **agregarPanelArbol()**.

```
 1 <?php
    $comportamientoVentana= new IgepPantalla();
```

```
    $panel = new IgepPanelArbol('PruebaArbol',"smty_datosPanel");
    $panel->activarModo("edi","estado_edi");
  5 $comportamientoVentana->agregarPanelArbol($panel);
    $s->display('p_pruebaArbol.tpl');
   ?>
```

4.1.2.1.3. Fichero de Plantillas (TPL)

Este fichero debe contener por un lado el árbol, y por otra parte, tantos paneles como tipos de nodos representables existan en el panel, los indicados con el método **setNodoPanel()**. Es importante tener en cuenta que se ha de mostrar sólo el panel activo, y para ello utilizamos la clausula *{if}* de Smarty, en la variable *$smty_panelVisible* nos vendrá el panel que queremos activo, es decir, nos vendrá el identificador que le habremos dado cuando hemos definido los nodos.

A continuación mostramos la tpl utilizada para el ejemplo:

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=$smty_descBreve textoAviso=$smty_textoAviso}
{CWBarra usuario=$smty_usuario codigo=$smty_codigo}
    {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
{/CWBarra}
{CWMarcoPanel conPestanyas="true"}
<!-- ****************** PANEL edi ***********************-->
{CWArbol estado=$estado_edi arbol=$smty_objArbol}
    {if $smty_panelVisible == "ANYOS" || $smty_panelVisible == "FACTURA-DG"}
        {CWPanel id="edi" tipoComprobacion="envio" itemSeleccionado=$smty_filaSeleccionada action="operarTabla" method="post" estado="on" claseManejadora="TinvFacturasArbol"
 accion=$smty_operacionFichaTinvFacturasArbol}
        {CWBarraSupPanel titulo=$smty_tituloPanel}
            {CWBotonTooltip imagen="01" titulo="Insertar registro" funcion="insertar" actuaSobre="ficha"}
            {CWBotonTooltip imagen="02" titulo="Modificar registro" funcion="modificar" actuaSobre="ficha"}
            {CWBotonTooltip imagen="03" titulo="Eliminar registro" funcion="eliminar" actuaSobre="ficha"}
            {CWBotonTooltip imagen="04" titulo="Limpiar registro" funcion="limpiar" actuaSobre="edicion"}
        {/CWBarraSupPanel}
        {CWContenedor}
            {CWFichaEdicion id="FichaEdicion" datos=$smty_datosPanel numPagInsertar="1"}
                {CWFicha}
                        {CWCampoTexto nombre="anyo" editable="true" size="4" value=$smty_anyoNuevo textoAsociado="A&ntilde;o"}
                          
                        {CWCampoTexto nombre="nfactura" editable="false" size="6" value ="0" textoAsociado="N&uacute;mero Factura"}
                        <br /><br />
                        {CWLista nombre="procedencia"  radio="true" editable="true" datos=$smty_datosPreInsertadosTinvFacturasArbol.procedencia textoAsociado="Procedencia"}
                        <br /><br />
                        {CWCampoTexto nombre="cif" editable="true" size="13" maxlength="13" textoAsociado="Cif"}
                         
                        {CWCampoTexto nombre="orden" editable="true" size="2" maxlength="2" textoAsociado="Orden" actualizaA="nombre"}
                         {CWCampoTexto nombre="nombre" editable="false" size="40"}
                        {CWBotonTooltip imagen="13" titulo="Ventana de seleccion" funcion="abrirVS" actuaSobre="cif" formActua="edi" panelActua="FichaEdicion"
 claseManejadora="TinvFacturasArbol"}
                        <br /><br />
                        {CWLista nombre="cdg" size="2" editable="true" datos=$smty_datosPreInsertadosTinvFacturasArbol.cdg textoAsociado="D.General"}
                        {CWCampoTexto nombre="fadquisicion" editable="true" size="10" conCalendario="true" textoAsociado="Fecha Adq"}
                        <br /><br />
                        {CWCampoTexto nombre="fcertificacion" editable="false" size="10" textoAsociado="Fecha Certif"}
                          
                        {CWCampoTexto nombre="codFactura" editable="true" size="13" maxlength="13" textoAsociado="Cod. Factura"}
                        </br></br>
                {/CWFicha}
                {CWPaginador enlacesVisibles="3"}
            {/CWFichaEdicion}
        {/CWContenedor}
        {CWBarraInfPanel}
            {CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar" vCamposComprobar=$smty_validacionesTinvFacturasArbol}
            {CWBoton imagen="42" texto=" Cancelar" class="boton" accion="guardar" action="IgepPruebaArbol__cancelarArbol"}
        {/CWBarraInfPanel}
    {/CWPanel}
  {else}
    {CWPanel id="edi" tipoComprobacion="envio" action="operarTabla" method="post" detalleDe="edi" estado=$estado_edi claseManejadora=""}
        {CWBarraSupPanel titulo=""}
        {/CWBarraSupPanel}
        {CWContenedor}
            {CWFichaEdicion  id="FichaDetalle" datos=$smty_datosPanel numPagInsertar="0"}
                {CWFicha}
                   </br></br></br></br>
                   NO HAY DATOS
                   </br></br></br></br>
                {/CWFicha}
            {/CWFichaEdicion}
        {/CWContenedor}
        {CWBarraInfPanel}
            {CWBoton imagen="42" texto=" Cancelar" class="boton" accion="guardar" action="IgepPruebaArbol__cancelarArbol"}
        {/CWBarraInfPanel}
    {/CWPanel}
  {/if}
{/CWArbol}
<!-- ****************** PESTAÑAS ***********************-->
    {CWContenedorPestanyas}
        {CWPestanya tipo="edi" estado=$estado_edi}
    {/CWContenedorPestanyas}
{/CWMarcoPanel}
{/CWVentana}
```

4.1.2.2. Otro ejemplo

A continuación se muestra un ejemplo en el que la definición de jerarquía está definida en un único campo (subconcepto) y en funcion de su longitud obtenemos el tipo de nodo. La jerarquía de nodos buscada es:

```
ANYOS
   ANYO
      CAP1    (longitud 1)
         CAP2    (longitud 2)
            CAP3    (longitud 3)
               CAP5   (longitud 5)
```

La forma de definir los nodos seria, partiendo de cada nodo raíz, ir definiendo como se obtienen los nodos hijos y así sucesivamente hasta tener definidos todos los nodos. Puesto que el nodo CAP5 no tiene hijos no necesitamos definir ese nodo.

```
...
$arbol->addNodoRaiz('ANYOS', 'A&ntilde;os de Subconceptos', 'SELECT',
        array('ANYO',"select concat(concat(concat(anyo,' ('),count(1)),')') as \"etiqueta\", anyo
                      from tcom_subconceptos_anyo group by anyo"));
$arbol->addNodoRama('ANYO',  'SELECT',
        array('CAP1',"SELECT anyo, subconcepto, concat(concat(subconcepto,' '),descrip) as \"etiqueta\"
                       FROM tcom_subconceptos_anyo WHERE anyo = %anyo% and length(subconcepto)=1
                       order by subconcepto", array("anyo")));
$arbol->addNodoRama('CAP1',  'SELECT',
        array('CAP2',"SELECT anyo, subconcepto, concat(concat(subconcepto,' '),descrip) as \"etiqueta\"
                      FROM tcom_subconceptos_anyo
                      WHERE anyo = %anyo% and substr(subconcepto,1,1)='%subconcepto%' and length(subconcepto)=2
                      order by anyo,subconcepto", array("anyo",'subconcepto')));
$arbol->addNodoRama('CAP2',  'SELECT',
        array('CAP3',"SELECT anyo, subconcepto, concat(concat(subconcepto,' '),descrip) as \"etiqueta\"
                      FROM tcom_subconceptos_anyo
                      WHERE anyo = %anyo% and substr(subconcepto,1,2)='%subconcepto%' and length(subconcepto)=3
                      order by anyo,subconcepto", array("anyo",'subconcepto')));
$arbol->addNodoRama('CAP3',  'SELECT',
        array('CAP5',"SELECT anyo, subconcepto, concat(concat(subconcepto,' '),descrip) as \"etiqueta\"
                      FROM tcom_subconceptos_anyo
                      WHERE anyo = %anyo% and substr(subconcepto,1,3)='%subconcepto%' and length(subconcepto)=5
                      order by anyo,subconcepto", array("anyo",'subconcepto')));
...
```

4.2. Componentes complejos

En este punto vamos a hablar de dos componentes que se salen de los básicos para diseñar una pantalla genérica y que nos ayudarán a la hora de trabajar con grupos de datos, tanto para seleccionar como para agrupar.

4.2.1. Ventana de selección

Las ventanas de selección es una particularización de las listas, siendo las siguientes características las que las diferencian:

- Se pueden filtrar los valores posibles mediante unas búsquedas sencillas. Lo que las hace aconsejables cuando el número de elementos posibles es superior a 100 aprox.
- Se puede visualizar más información complementaria que nos ayude a la selección de un registro, es decir, en la ventana de selección se mostrarán tantas columnas como se necesiten para la mejor comprensión del registro.
- Del registro seleccionado podemos recuperar otra información que nos interese para llevárnosla a otros campos del panel origen. Esta información puede ser visible o no en la ventana de selección.
- Nos permite tener diferentes fuentes de datos, los datos pueden venir de un web service, de base de datos, de una clase...
- Se puede crear una ventana de selección de imágenes.

4.2.1.1. Definición de la ventana

Para definir una ventana de selección tenemos que definirla en la clase manejadora. Creamos una instancia de la clase **gvHidraSelectionWindow()** pasándole el nombre de campo de la tpl, campo del que dependerá la ventana de selección, y como segundo parámetro, un identificador (p.ej. USUARIOS) del origen de los datos, opcionalmente, podemos indicar una conexión alternativa como tercer parámetro, para ello debemos indicar pasar un DSN de conexión. Después tendremos que ir relacionando los campos del panel que van a ser actualizados con el campo correspondiente de la base de datos, con el método **addMatching()**, como mínimo habrá una relación ya que de lo contrario no

actualizaríamos ningún campo de la ventana orígen. Por último añadimos esta definición de la ventana al panel con **addSelectionWindow()**.

```
$usuarios = new gvHidraSelectionWindow('filUsuario','USUARIOS');
$usuarios->addMatching('filId','id');
$usuarios->addMatching('filUsuario','usuario');
$usuarios->addMatching('filNombre','nombre');
$this->addSelectionWindow($usuarios);
```

Además de esta definición básica la ventana tiene dos métodos que nos permitirán customizarla:

- **setSize()**: este método permite fijar el tamaño de la ventana emergente.

```
$usuarios = new gvHidraSelectionWindow('usuario','USUARIOS');
...
$usuarios->setSize(800,600);
...
```

- **setTemplate()**: nos permite diseñar nuestra propio contenido de la ventana de selección, es decir, tendrá una tpl particular en la que podremos seleccionar el tipo de componente (CampoTexto, List, Imagen,), el nombre de la columna, el ancho de las mismas, ...

```
$usuarios = new gvHidraSelectionWindow('usuario','USUARIOS');
...
$usuarios->setTemplate('ventanaSeleccion/incUsuarios.tpl');
...
```

Si utilizamos este método significa que tenemos una plantilla (tpl) que dará formato a la ventana de selección. En caso contrario, el framework dibuja una ventana de selección sencilla, en la que su contenido no es configurable.

Siguiendo el caso del ejemplo, debemos tener un directorio dentro del directorio plantillas, llamado "ventanaSeleccion", y dentro tendremos las tpl que correspondan a las ventanas de selección de la aplicación. En esa plantilla se crearán los campos que queremos que nos sean necesarios, tanto visibles como ocultos.

```
{CWImagen nombre="fichero" rutaAbs="yes" textoAsociado="Fichero" width="40" height="50"}
{CWCampoTexto nombre="nombre" size="15" textoAsociado="Nombre"}
{CWCampoTexto nombre="descripcion" size="40" textoAsociado="Descripción"}
{CWCampoTexto nombre="id" oculto="true"}
```

Una vez definida la ventana sólo nos queda introducir en la tpl un componente CWBotonTooltip que haga referencia a una ventana de selección. Para ello se parametriza indicando el campo sobre el que actua (en nuestro ejemplo usuario), el form y el panel. Aqui tenemos un ejemplo:

```
{CWCampoTexto nombre="filUsuarios" size="8" textoAsociado="Usuario"}
{CWBotonTooltip imagen="13" titulo="Busqueda de Usuarios" funcion="abrirVS" actuaSobre="filUsuarios"
 formActua="fil" panelActua="fil" claseManejadora="TpqmVehiculosPorUsuario"}
```

4.2.1.2. Fuentes de datos

Las definiciones particulares a la aplicación, se realizarán, al igual que en el caso de las listas, se definirán en el arranque de la aplicación. Para ello debemos de añadir su definición en la clase que controle el panel principal de la aplicación, fichero **AppMainWindow.php**. Ya que el origen de los datos puede ser diferente, tenemos dos formas de indicarlo:

- **setSelectionWindow_DBSource($key, $query, $fields=null)**:

  Método para introducir fuente de datos con consulta SQL.

  El primer parámetro será el identificador que nos es necesario cuando definimos la ventana en la clase manejadora. El segundo parámetro es la propia consulta que nos devolverá los datos a mostrar. Una condición que hay que tener en cuenta a la hora de definir esta consulta, es que el parámetro, o parámetros, que nos interesa que se devuelvan a la ventana tengan el mismo nombre que en la ventana orígen, para ello podemos utilizar alias en la select, estos alias serán los que aparezcan en la cabecera de la columna. El tercer parámetro, opcional, nos permite añadir campos a los que ya aparecen en la consulta, sobre ellos también se aplicará el filtro de búsqueda.

  En la consulta también podemos usar subconsultas. En el caso de postgres, estas han de tener siempre alias en el from (aunque no se vaya a usar) por lo que conviene ponerlo siempre (para cualquier tipo de base de datos). Notar también

que en la subconsulta sólo pondremos alias para los campos que queramos renombrar, no afectan las mayúsculas / minúsculas.

```
$conf->setSelectionWindow_DBSource('CENTROS','select ccentro as '\lisCcentro\', dcentro as '\lisDcentro\' from tinv_centros');
```

- **setSelectionWindow_ClassSource($key, $query)**:

  Método para introducir fuente de datos con una clase. Esta clase debe implementar la interfaz **gvHidraSelectionWindow_Source.php**.

```
$conf->setSelectionWindow_ClassSource('PERSONAS',"PersonasSource");
```

4.2.1.3. Búsqueda en la ventana

La búsqueda, por defecto, busca el texto introducido en cualquiera de los campos que se visualizan, y si obtenemos los datos de una consulta a bd también buscará en el array de campos añadidos. Podemos cambiar este comportamiento de forma similar a los filtros en paneles. Veamos un ejemplo:

```
$usuarios = new gvHidraSelectionWindow('filUsuario','USUARIOS');
$usuarios->addMatching('filId','id');
$usuarios->addMatching('filUsuario','usuario');
$usuarios->addMatching('filNombre','nombre');
$usuarios->setQueryMode(2); // valor por defecto es 1
$this->addSelectionWindow($usuarios);
```

Por defecto, cuando el usuario introduce un texto para buscar, busca por todos los campos que aparecen en la consulta concatenados y separados por un espacio: concat(concat(col1,' '),col2)... De esta forma el usuario puede hacer una búsqueda que coincida con el final de un campo y el principio del siguiente. Es importante que para las dos fuentes de datos, los parámetros del matching son los que recogerán los valores y los introducirán en la ventana destino.

4.2.1.4. Dependencia

Al igual que en las listas, tenemos la posibilidad de crear ventanas de selección dependientes de otros controles, aunque, en este caso, podemos definir dos tipos de dependencia (fuerte o débil). La fuerte añade a la WHERE de la consulta que se genere en la ventana una clausula con el valor del campo del que depende sin tener en cuenta si este tiene valor o no. La débil, sólo lo hace si el campo tiene valor. Para indicar la dependencia tenemos que hacer uso del método setDependence indicando los campos de la tpl, su correspondencia en la BD y el tipo de dependencia deseado :0-> fuerte(valor por defecto) o 1-> débil. En nuestro ejemplo, sería:

```
$usuarios = new gvHidraSelectionWindow('filUsuario','USUARIOS');
$usuarios->addMatching('filId','id');
$usuarios->addMatching('filUsuario','usuario');
$usuarios->addMatching('filNombre','nombre');
$usuarios->setDependence(array('esDePersonal'),array('tper_ocupacion.esPersonal'),0);
$this->addSelectionWindow($usuarios);
```

4.2.2. Selector

El selector corresponde con el plugin CWSelector, es un plugin que aún está poco elaborado, se creó para representar relaciones 1:N dentro de un mismo panel. De todos modos, la implementación es manual, es decir. el programador será el que realice las operaciones.

Aquí vemos el aspecto que nos dará el selector en una pantalla.

Vamos a explicar como hacer uso de él:

- *En la plantilla (tpl)*:

  El plugin CWSelector solamente aparecerá dentro de una ficha (CWFicha). CWSelector es un plugin block, dentro del cual podremos definir otros componentes, como CWCampoTexto y CWLista.

```
{CWSelector titulo="TituloDelGrupo" botones=$smty_botones nombre="listaBienes" editable="true" separador="."}
    {CWCampoTexto nombre="CampoTexto1" editable="false" size="6" textoAsociado="CampoTexto1"}
    {CWCampoTexto nombre="CampoTexto2" editable="false" size="6" textoAsociado="CampoTexto1"}
    {CWCampoTexto nombre="CampoTexto3" editable="false" size="6" textoAsociado="CampoTexto1"}
{/CWSelector}
```

  Los valores introducidos en estos campos serán copiados, cuando pulsamos el botón , a la lista múltiple que tenemos en la parte izquierda, separando los valores por el caracter introducido en el parámetro *separador*, en nuestro ejemplo el punto.

  Con el botón , copiaremos los valores que estén seleccionados en la lista múltiple a los campos que hayamos definido dentro del selector.

  Finalmente, el botón nos servirá para eliminar el registro que tengamos seleccionado en la lista múltiple.

- *En el views*:

  En el views tenemos que indicar qué botones del selector tendremos activos, para ello asignamos la variable smarty *smty_botones*.

```
$botones = array('insertar','modificar','eliminar');
$s->assign("smty_botones",$botones);
```

- *En la clase manejadora*:

  Finalmente, ¿como recuperamos el contenido del Selector en la clase manejadora? Recordemos que el contenido del selector devolverá una lista, ya que podemos tener N items, pero al recibirlo en la clase manejadora, sólo recibimos un string de elementos separados por separador. Por esta razón, para obtener un array podemos utilizar el siguiente código:

```
public function postInsertar($objDatos) {

    $m_datos = $objDatos->getAllTuplas();
    foreach($m_datos as $tupla) {
       if($tupla['listaBienes']!='') {
           echo 'El resultado es: ';
           print_r($tupla['listaBienes']);die;
       }
    ...
}
```

  Si hubiéramos introducido los siguientes valores en nuestro ejemplo:

  El resultado sería:

```
Array
(
    [0] => 1.2.3
    [1] => 4.5.6
)
```

4.3. Tratamiento de ficheros

Vamos a contar como trabajar con documentos y ficheros de cualquier tipo. Añadimos ejemplos de como insertar un fichero en la BD, como mostrar una imagen que se encuentra en el servidor en un campo CWImagen, como insertar N tuplas en una BD a partir de un fichero XML.

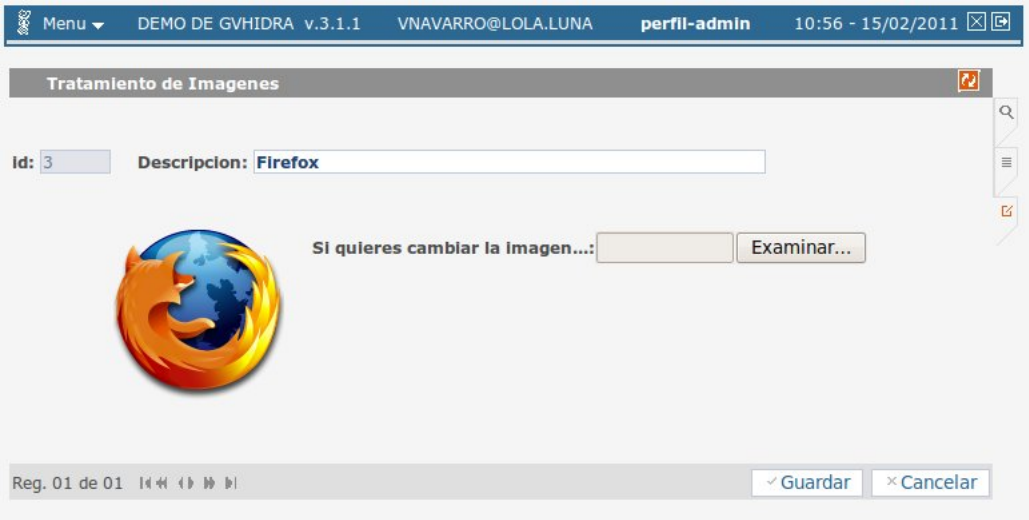
4.3.1. Manejo de ficheros de imágenes

En este primer ejemplo vamos a mostrar como crear un mantenimiento que almacene imágenes en la BD (en nuestro ejemplo PostgreSQL) y las muestre en un **CWImagen**. Para conseguir esto necesitamos que en la interfaz (en la tpl) aparezcan dos plugins especiales de tratamiento de ficheros (**CWUpLoad** y el **CWImagen**).

El primero de ellos, el **CWUpLoad**, nos permite seleccionar un fichero ubicado en el ordenador cliente y subirlo al servidor Web. El contenido de este fichero se almacena en una ubicación temporal y el programador es el encargado de hacer uso de él. Posteriormente mostraremos los métodos que gvHidra proporciona para que se tenga acceso a la información de dicho fichero.

El segundo de ellos es el **CWImagen**, que permite mostrar una imagen en un panel a partir de una URL.

A continuación mostramos un ejemplo del panel resultado:



Para conseguir esto tenemos que tener en cuenta las dos partes que van a suponer trabajo para el programador, por un lado el tratamiento del fichero imagen cuando el usuario lo selecciona con el **CWUpLoad** y lo quiere almacenar en el servidor y, por otro, la representación de un fichero imagen en el **CWImagen**. El primero de estos pasos necesita de un acceso a la información del fichero y la inserción del mismo en la tupla que se esté modificando. A continuación mostramos el código que se ha insertado en el panel para resolverlo.

```
public function postModificar($objDatos)
{
    do
    {
      $datosFile = $objDatos->getFileInfo('ficheroUpload');
      if(is_array($datosFile) and ($datosFile['tmp_name']!='')){
        $fichero = pg_escape_bytea(file_get_contents($datosFile['tmp_name']));
        $res = $this->operar("update tinv_imagenes SET fichero='{$fichero}', nombre='".$datosFile['name']."' where id=".$objDatos->getValue('id'));
        if($res==-1)
        {
          $this->showMensaje('APL-25');
          return -1;
        }
      }
    } while($objDatos->nextTupla());
    return 0;
  }
```

De este código destaca, en primer lugar, el método **getFileInfo()** del objeto objDatos. Este método devuelve el array de información del HTTP que indica los datos del fichero que ha subido el usuario (nombre, tipo, ubicación,...). A partir de esta información el programador ya puede hacer uso del fichero; y, en este caso, decide insertarlo en una BD PostgreSQL. Para ello va a hacer uso de un campo de tipo Bytea y, por esta razón, utiliza las siguientes premisas. Primero, tras obtener el contenido del fichero (función php *file_get_contents*) utiliza la función nativa de php *pg_escape_bytea* para poder construir la consulta. Construye la consulta incluyendo el contenido (escapado) del fichero entre llaves. De este modo la instrucción no tiene problemas al pasar el array de bytes al PosgreSQL. El otro punto complejo de esta pantalla

está en la recuperación de la imagen de la BD. Para ello debemos obtener la imagen del campo bytea y almacenarla en una ubicación accesible por nuestro plugin **CWImagen**. Por esta razón, en este caso hemos decidido crear un directorio imagenes en la raiz de nuestro proyecto (también se podria usar el templates_c que ya usamos para smarty) donde se crearán los ficheros que se quieran visualizar. Luego se asignará al **CWImagen** la ruta de cada uno de ellos. El código que se encarga de realizar esto es:

```
public function postEditar($objDatos)
{
    //Comprobamos si tiene imagen y en este caso la cargamos.
    $res = $objDatos->getAllTuplas();
    foreach($res as $indice => $tupla)
    {
      if($tupla['fichero']!='')
      {
        $nuevoFichero = "imagenes/".$tupla['nombre'];
        if(!file_exists($nuevoFichero))
        {
          $imagen = pg_unescape_bytea($tupla['fichero']);
          $gestor = fopen($nuevoFichero, 'x');
          fwrite($gestor,$imagen);
        }
        $res[$indice]['fichero'] = $nuevoFichero;
      }
    }
    $objDatos->setAllTuplas($res);
}//Function postEditar
```

Como se puede suponer del código anterior, el CWImagen se llama fichero.

```
{CWImagen nombre="fichero"}
{CWUpLoad nombre="ficheroUpload" size="10" textoAsociado="Si quieres cambiar la imagen..."}
```

4.3.2. Manejo de ficheros de cualquier tipo

Puede darse el caso que queramos almacenar documentos de cualquier tipo en la BD. Para ello tendríamos que hacer una simple modificación al código anterior. Es evidente que ya no nocesitariamos el plugin CWImagen. Lo único que necesitamos es un botón que lance una acción particular que recupera el documento de la BD y lance el mismo a una nueva ventana del navegador. El código que se encarga de realizar esto es el siguiente

```
public function accionesParticulares($str_accion, $objDatos)
{
    switch ($str_accion)
    {
      case 'verFichero':
        //Bucle para crear un listado para cada peticion seleccionada:
        $objDatos->setOperacion("seleccionar");
        $m_datosSeleccionados = $objDatos->getAllTuplas();
        $res = $this->consultar("SELECT fichero,tipo from tinv_documentos WHERE id = ".$m_datosSeleccionados[0]['id']);
        if($res[0]['fichero']!='')
        {
          header('Content-Type: '.$res[0]['tipo']);
          header('Content-Disposition: attachment; filename="'.$m_datosSeleccionados[0]['nombre'].'"');
          print(pg_unescape_bytea($res[0]['fichero']));
          ob_end_flush ();
          //Para que no continue la ejecución de la página
          die;
        }
        $actionForward = $objDatos->getForward('correcto');
       break;
    }
    return $actionForward;
}
```

Como nota podemos indicar que, a diferencia del ejemplo anterior, en este ejemplo, al almacenar el documento en la BD se almacena no sólo su contenido, sino también el tipo. De este modo se puede asignar al header para que el navegador lo abra con el programa adecuado para tratarlo dependiendo de la extensión (MIME-TYPES).

4.3.3. Importar datos a la BD desde fichero

Este es otro de los ejemplos típicos con los que nos podemos encontrar. Imaginemos el caso de una aplicación que carga sus datos a partir de una fuente externa como, por ejemplo, un XML. Para este ejemplo sólo utilizaremos el plugin CWUpLoad y el código sería el siguiente:

```
public function accionesParticulares($str_accion, $objDatos)
{
    switch ($str_accion)
    {
      case 'importarFichero':
        $objDatos->setOperacion('external');
```

```
      $datosFile = $objDatos->getFileInfo('ficheroUpload');
      $contenidoFichero = utf8_encode(file_get_contents($datosFile['tmp_name']));
      $dom = domxml_open_mem($contenidoFichero,
        DOMXML_LOAD_PARSING + //0
        DOMXML_LOAD_COMPLETE_ATTRS + //8
        DOMXML_LOAD_SUBSTITUTE_ENTITIES + //4
        DOMXML_LOAD_DONT_KEEP_BLANKS //16
      );
      if(is_object($dom))
      {
        $xpathXML = $dom->xpath_new_context();
        $xpresultXML = $xpathXML->xpath_eval("//persona", $xpathXML);
        $v_personas = $xpresultXML->nodeset;
        //preparamos los contadores
        $insertados = 0;
        $totales = 0;
        foreach($v_personas as $persona)
        {
          $nif = $persona->get_attribute('nif');
          $comprobacion = $this->consultar('SELECT count(1) as "cuenta" FROM tinv_xml where nif=\''.$nif.'\'');
          if($comprobacion[0]["cuenta"]==0)
          {
            $nombre = $persona->get_attribute('nombre');
            $apellidos = $persona->get_attribute('apellidos');
            $correcto = $this->operar("INSERT INTO tinv_xml values ('".$nif."','".$nombre."','".$apellidos."')");
            if($correcto!=-1)
              $insertados++;
          }
          $totales++;
        }
        $this->showMensaje('APL-26',array($insertados,$totales));
      }
      else {
        $this->showMensaje('APL-27');
      }
      $this->setResultadoBusqueda(array());
      $actionForward = $objDatos->getForward('correcto');
      break;
    default:
      die("La acción $str_accion no está activada para esta clase.");
  }
  return $actionForward;
}
```

Como podemos ver, la aplicación espera un fichero XML del que extrae todos los tags <persona>. Si no existen en la BD los inserta con los atributos deseados.

4.4. Control de la Navegación. Saltando entre ventanas

Uno de los típicos problemas en el diseño de aplicaciones web es el control de la navegación. Para ayudar en la resolución de este problema, el framework incorpora un mecanismo que facilita el uso de estos saltos entre páginas con el consiguiente paso de mensajes.

Para entender la problemática que implica los saltos entre páginas, vamos a destacar ciertos puntos que se deben considerar:

- **Paso de parámetros**: evidentemente, se debe poder pasar información desde el origen hasta el destino. En nuestro caso desde una clase manejadora a otra. Para ello, tenemos que hacer uso de la SESSION.
- **Pila de saltos**: de alguna manera, una vez volvamos de un salto debemos limpiar todo rastro del salto, para evitar colisiones con futuras operaciones.
- **Automatización del proceso**: intentar que el proceso sea lo más limpio posible para el programador.

Teniendo en cuenta estas consideraciones, el framework ha implementado una clase y una serie de métodos con la intención de facilitar el proceso al programador. Hay que destacar que un salto puede ser tan simple como indicar una URL en el actionForward correspondiente; pero con ello perderiamos todo control sobre el retorno (no podríamos garantizar la "limpieza" de la SESSION).

4.4.1. Implementación

Vamos a implementar un ejemplo. Tenemos un mantenimiento tabular en el que vamos a acumular fila a fila una serie de aspirantes a los que les queremos realizar una entrevista grupal. Estos aspirantes se obtienen a través de otra ventana

(también tabular) en la que podemos buscar los aspirantes por sus perfiles y seleccionarlos para la entrevista. Una vez seleccionados los aspirantes deseados, se pulsa al botón "Citar para entrevista" disparando el proceso (envío de correos, inserción en el planning del entrevistador, ...).

Ficheros implicados:

Tabla 4.1. Ficheros implicados en implementación

clase manejadora	view	plantilla tpl	descripción
GeneradorEntrevistas.php	p_generadorEntrevistas.php	p_generadorEntrevistas.tpl	Generador de entrevistas. Origen del salto
Aspirantes.php	p_aspirantes.php	p_aspirantes.tpl	Mantenimiento de aspirantes.

Dando por hecho la creación de los dos mantenimientos vamos a ir, paso a paso, indicando como implementar el salto. En primer lugar, tenemos que ubicar un punto de disparo de la acción de salto. Puede ser bien un botón clásico:

```
{CWBoton id="saltoVisualizar" imagen="49" texto="SALTO" class="boton" accion="saltar"}
```

o un botón tooltip:

```
{CWBotonTooltip imagen="39" titulo="+" funcion="saltar" actuaSobre="ficha"}
```

Ahora, tenemos que crear en la clase manejadora origen del salto (GeneradorEntrevistas), un método que gestione el salto (si se debe saltar o no; si pasa parámetros). En este caso el framework proporciona un método para tal uso **saltoDeVentana**:

```
public function saltoDeVentana($objDatos, $objSalto){
    //Nombre de la claseM destino del salto
    $objSalto->setClase('Aspirantes');
    //Nombre de la accion de entrada a la ventana
    $objSalto->setAccion('iniciarVentana');
    //Nombre de la accion de retorno
    $objSalto->setAccionRetorno('iniciarVentana');
    //Parametro: numero maximo de seleccionables
    $objSalto->setParam('numeroMaxSeleccionable',5);
    return 0;
}
```

Como se puede ver, por un lado tenemos los datos relativos a la pantalla (objDatos) y por otro, los relativos al salto. El objeto salto es el que se deberá configurar para indicar la clase destino del salto, **setClase()**, y la acción de entrada en la nueva ventana (el mapeo correspondiente), **setAccion()**, también se estable la acción de retorno a la ventana origen con **setAccionRetorno()**. También se pueden añadir parámetros al salto usando el método **setParam()**.

Este código, redireccionará la ejecución hacia la clase *Aspirantes* entrando con la acción *iniciarVentana*. Esto implica que en el fichero de mapeos (mappings.php) tenemos que definir el _AddMapping y los _AddForwards correspondientes.

Ahora, en la clase destino del salto, debemos recoger la información transmitida en el salto. Para ello, en el constructor añadiremos el siguiente código:

```
//Comprobamos si hay salto
$objSalto = IgepSession::damePanel('saltoIgep');
if(is_object($objSalto)){
    $this->parametros = $objSalto->getParams();
}
```

Tenemos que acceder a la zona de la SESSION donde el framework almacena la información del salto y recuperar el objeto.

Una vez realizado el salto, vamos a indicar como se realiza el retorno. Para ello primero vamos a ver en la tpl como indicar que vamos a lanzar una acción de retorno. En nuestro caso hemos añadido dos botones que controlan el retorno, uno acumula los seleccionados, el otro cancela la operación. En ambos casos la acción es volver, los diferenciamos a través del id.

```
{CWBoton id="acumular" imagen="49" texto="ACUMULAR" class="boton" accion="volver"}
{CWBoton id="cancelarAcumular" imagen="42" texto="CANCELAR" class="boton" accion="volver"}
```

Este estímulo se recibirá en la clase objeto del salto (en nuestro ejemplo Aspirantes). En ella debemos sobre escribir el método regresoAVentana:

```
public function regresoAVentana($objDatos, $objSalto){
    if($objSalto->getId()=='acumular'){
        //Recogemos los datos seleccionados por el usuario en pantalla
        $m_datos = $objDatos->getAllTuplas('seleccionar');
        $objSalto->setParam('nuevosAspirantes',$m_datos);
        $objSalto->setAccionRetorno('acumular');
    }
    return 0;
}
```

En nuestro ejemplo, en el caso de que el usuario haya seleccionado aspirantes y haya pulsado acumular, volveremos a la clase con la acción acumular. En caso contrario volveremos con la acción que se programó en el salto (iniciarVentana). Por supuesto, la clase origen del salto, tiene que tener en el fichero de mapeo el *_AddMapping* y los *_AddForwards* correspondientes (en nuestro caso GeneradorEntrevistas__acumular y GeneradorEntrevistas__iniciarVentana).

Finalmente, en la clase origen del salto, tenemos que configurar el regreso del mismo. En el caso de un regreso por cancelación de acción, no tenemos que hacer nada, ya que se trata de una acción genérica del framework (iniciarVentana). En el caso de un regreso para acumular, se ha decidido realizar una acción no genérica: una acción particular. Se trata de la acción GeneradorEntrevistas__acumular.

Para tratarla tenemos que incorporar el siguiente código.

```
public function accionesParticulares($accion, $objDatos) {
    switch ($accion) {
        case 'acumular':
            $salto = IgepSession::damePanel('saltoIgep');
            $nuevos = $salto->getParam('nuevosAspirantes');
            $this->acumularAspirantes($nuevos);
            $actionForward = $objDatos->getForward('correcto');
            break;
        }
        return $actionForward;
}
```

En el método acumularAspirantes, se realizarán las acciones pertinentes según la lógica del caso de uso. En nuestro ejemplo tendriamos que añadir los nuevos aspirantes a los seleccionados previamente:

```
public function acumularAspirantes($nuevos)
{
   $acumulados = $this->getResultadoBusqueda();
   if($acumulados == '')
      $acumulados = array();
   foreach($nuevos as $indice =>$linea)
   {
      $aspirante = array();
      $aspirante["dni"] = $linea["dni"];
      $aspirante["nombre"] = $linea["nombre"];
      ...
      array_push($acumulados,$aspirante);
   }
   $this->setResultadoBusqueda($acumulados);

   // si en vez de mostrar los aspirantes seleccionados quisieramos modificarlos
   // directamente en la base de datos, luego podriamos refrescar la pantalla con:
   //$this->refreshSearch();
}
```

Con esto ya tenemos un ejemplo entero de saltos. Como antes indicábamos, en futuras versiones este mecanismo se revisará y mejorará.

4.5. Carga dinámica de clases

4.5.1. Introducción

Hasta ahora todos los ficheros que usamos en una aplicación se cargan siempre independientemente de que se usen o no en el hilo actual de ejecución. Puesto que normalmente no se usa más de una clase de actions simultáneamente, esta situación se puede mejorar si cargamos las clases dinámicamente, y puede notarse especialmente en las aplicaciones grandes.

Desde la versión 5 de PHP, existe una funcionalidad nueva que permite, cuando no se encuentra una clase, llamar a una función de usuario donde se incluya el fichero correspondiente. Aprovechando esta funcionalidad nueva, se ha creado una clase en el framework que permite:

- **registerClass**: indicar cada clase en que fichero se ubica
- **registerFolder**: permite almacenar carpetas en las que buscar las clases (siempre que su nombre coincida con un fichero con extensión php)

Combinando los dos métodos podemos configurar la carga de clases de la manera más cómoda y eficiente teniendo en cuenta que:

- por defecto se registra la carpeta '*actions*'
- si una clase no se llama igual que el fichero que la contiene es necesario usar el método '*registerClass*'.

En cualquier caso, si no se desea utilizar esta funcionalidad basta con no invocar ninguno de los métodos anteriores. En el fichero include.php de la aplicación podemos seguir haciendo los includes de la manera tradicional.

4.5.2. Ejemplos de utilización

```
// obtenemos referencia al objeto
$al = GVHAutoLoad::singleton();

// registramos una carpeta donde tenemos clases
$al->registerFolder('actions/listados');

// si tengo pocas clases en una carpeta puedo optar por registrarlas individualmente
$al->registerClass('cabFactura', 'actions/factura/cabFactura.php');
$al->registerClass('linFactura', 'actions/factura/linFactura.php');

// si las clases no se llaman igual que el fichero donde se encuentran,
// no hay mas remedio que registrarlar individualmente
$al->registerClass('TinvTipos2',    'actions/TinvTipos.php');
$al->registerClass('TinvSubtipos2', 'actions/TinvSubtipos.php');

...
```

Parte III. Complementos al desarrollo

Tabla de contenidos

Capítulo 5. Fuentes de datos

5.1. Conexiones BBDD

5.1.1. Bases de Datos accesibles por gvHidra

Una de las características importantes que tenemos en PHP es la gran cantidad de distintos tipos de bases de datos soportados. Cada una de ellas requiere de una extensión que define su propia API para realizar la misma tarea, desde establecer una conexión hasta usar sentencias preparadas o control de errores. Cambiar las llamadas a la API cuando cambiamos de BD en una aplicación es costoso pero se puede hacer, aunque en el caso de un framework no es factible porque tiene que funcionar el mismo código con varias BD. Lo que se suele hacer es añadir una capa de abstracción que nos permita operar de la misma manera con todos los tipos de BD (al menos para las operaciones más habituales o las más estandarizadas). En el caso de gvHIDRA se utiliza la librería MDB2 de PEAR (http://pear.php.net/package/MDB2/), una de las más usadas para este fin gracias a su amplia funcionalidad y a la actividad del proyecto. De esta forma podemos usar una BD sin preocuparnos por el tipo, ya que tenemos una API única.

Sin embargo, a pesar de usar la capa de abstracción, sigue habiendo aspectos que dependen del tipo de BD: uso de características especiales (secuencias, limit, bloqueos, ...), manejo de transacciones, representación de fechas y números, ... Para resolver estos problemas, definimos en el framework otra capa de abstracción (que se implementa en unas clases que se ven en el apartado siguiente) y las funciones más importantes que realizan son:

- Definición del formato de fechas y números que se va a usar para comunicarse con el gestor de BD. Este formato es transparente para el programador, ya que es el framework el que hace las conversiones necesarias cuando enviamos/recogemos información a/de la BD. Normalmente nos referiremos a éste como **formato de BD**.

- Inicialización de la conexión: son operaciones que se hacen cada vez que se hace una nueva conexión. Por ejemplo se fija la codificación que se va a usar en la comunicación (actualmente LATIN1 o ISO-8859-1, que es la usada en el framework), o se modifican parámetros de la conexión para establecer el formato de BD.

- Ajustes por sintaxis: aquí se incluyen una serie de métodos para obtener la sintaxis de ciertas operaciones que se pueden comportar de forma distinta en cada BD. Por ejemplo la clausula limit (para paginar resultados de consultas), función para concatenar cadenas, uso de secuencias, ...

- Unificar tratamiento de errores: por ejemplo, cuando hay un registro bloqueado, poderlo detectar independientemente de la BD.

Actualmente las BD soportadas por gvHIDRA son postgresql, mysql y oracle. Conforme vayamos soportando nuevos tipos (entre los soportados por MDB2), la funcionalidad de esta capa se podrá ir aumentando.

5.1.2. Acceso y conexión con la Base de Datos

Cómo crear conexiones a distintas fuentes de datos.

5.1.2.1. Introducción

El framework ofrece una serie de facilidades al programador que pueden llegar a ocultar las posibilidades que tiene de conexión a BBDD. Concretamente, tanto el pattern gvHidraForm_DB o el debug (IgepDebug), son ejemplos de uso de conexiones de forma transparente para el programador. En ambos casos, gvHidra se encarga de conectar con el SGBD correspondiente y operar con él, dejando una serie de métodos que para consultar/operar.

Evidentemente, puede que nos encontremos con circunstancias que nos obliguen a utilizar algo más de lo que hemos expuesto. Para cubrir todo ello, el framework ofrece la clase IgepConexion que, a través de la librería PEAR::MDB2 conecta con diferentes SGBD. Esto permite crear conexiones a diferentes fuentes de forma homogenea e independizárnos de los drivers nativos que trabajan por debajo.

A esta capa de abstracción, el framework aporta una serie de perfiles de SGBD (IgepDBMS) que nos permiten configurar dichas conexiones para, por ejemplo, independizar el formato de la configuración del SGBD. Actualmente se han definido los siguientes perfiles:

Tabla 5.1. Perfiles SGBD

SGBD	PERFIL (CLASE)
Oracle	IgepDBMS_oci8.php
PostgreSql	IgepDBMS_pgsql.php
MySQL	IgepDBMS_mysql.php

Nota

se pueden crear perfiles para otros SGBD de forma sencilla.

5.1.2.2. Creación de una conexión

Para crear una conexión a una base de datos simplemente tenemos que crear una instancia de la clase IgepConexion pasándole el dsn de conexión a dicha BBDD. Dicho dsn tiene que tener la siguiente estructura (en PHP):

```
//Ejemplo postgresql
$dsnPos = array(
    'phptype'  => 'pgsql',
    'username' => 'xxxx',
    'password' => 'xxxx',
    'hostspec' => 'xxxx',
    'database' => 'xxxx',
);

//Ejemplo oracle
$dsnOra = array(
    'phptype'  => 'oci8',
    'username' => 'xxxx',
    'password' => 'xxxx',
    'hostspec' => 'xxxx',
);
```

Estos dsn se tienen que definir en el fichero de configuración de la aplicación gvHidraConfig.inc.xml siguiendo la sintaxis que especifica su DTD. También se pueden definir en la carga dinámica, aunque es poco recomendable.

La construcción de la conexión sería tan simple como:

```
//Nueva conexion a postgresql
$conPos = new IgepConexion($dsnPos);

//Nueva conexion a oracle
$conOra = new IgepConexion($dsnOra);
```

Después de crear la instancia, hay que comprobar si ha habido problemas de conexión:

```
if (PEAR::isError($conPos->getPEARConnection()))
     return;
```

¿Conexiones persistentes?

Si, conexiones persistentes aunque con matizaciones. El driver nativo de PHP para PostgreSQL ofrece la posibilidad de, con el objeto de optimizar, reutilizar las conexiones. De modo que, si creamos 10 conexiones, realmente tenemos 10 referencias a la misma conexión. Esto aumenta el rendimiento ya que sólo tenemos un "coste" de conexión aunque, evidentemente es peligroso. Por ello el framework, por defecto, crea conexiones independientes que mueren cuando el programador destruye el objeto (o cuando acaba el hilo de ejecución).

Puede que, en algunos casos puntuales, se quiera utilizar este tipo de conexiones (sólo funcionarían en PostgreSQL), y para esos casos se ha creado un parámetro en el constructor de la conexion (bool). Este parámetro, persistent, indica si el programador quiere solicitar una conexion persistente, siendo su valor por defecto false.

```
//Conexion permanente (reusable/reusada)
$pcon = new IgepConexion($dsn_log,true);
```

5.1.2.3. Utilizar conexión de la clase manejadora (gvHidraForm_DB)

Puede que en ciertas circunstancias nosotros queramos recuperar/utilizar la conexión que mantiene automáticamente la clase manejadora (las que extienden de gvHidraForm_DB). Para ello, se facilita el método getConnection que devuelve la instancia de la conexión en uso.

```
//En la clase manejadora
$con = $this->getConnection();
```

Nota

la clase manejadora ofrece también los métodos consultar y operar sobre esta conexión para poder lanzarlos sin recuperar la instancia de la conexión.

5.1.2.4. Métodos de conexión

En este apartado vamos a ver algunos de los métodos interesantes que ofrece la clase de conexiones de gvHIDRA. No los vermos todos (para ello consultar PHPDoc o doxy), pero si los más útiles.

5.1.2.4.1. consultar

Este método sirve para realizar una query con la conexión activa. El primer parámetro corresponde a la SELECT que queremos ejecutar. Opcionalmente podemos pasarle un segundo parámetro indicando si queremos que transforme los datos (especialmente fechas y números) al formato interno del framework. Sino se especifica este segundo parámetro, los datos vendrán en el formato configurado para cada SGBD, por lo que puede que no sean adecuados para operar con ellos.

```
// obtenemos datos para operar en capa de negocio (FW)
$res=$con->consultar(" SELECT fecha_ini,fecha_fin
            FROM tper_permisos
            WHERE nregpgv = '44908412R'",
                    array( 'DATATYPES'=>array('fecha_ini'=>TIPO_FECHA,'fecha_fin'=>TIPO_FECHA,)));
```

5.1.2.4.2. operar

Este método sirve para lanzar operaciones DML más allá del comando SELECT; es decir UPDATE,INSERT o DELETE. En ocasiones necesita que se ejecute sobre los datos el método prepararOperacion.

5.1.2.4.3. prepararOperacion

Convierte los campos al formato que acepta la base de datos (escapar los carácteres especiales en textos, ajustar los separadores decimales y de grupos en números y formatos en fechas). Hay que llamarla antes del método consultar, operar o preparedQuery. Los parámetros que acepta son:

1. valor: puede ser un valor simple o un array de registros, donde cada registro es un array con estructura 'campo'=>valor. Es un parámetro por referencia, por lo que devuelve el resultado en esta variable.

2. tipo: si el valor es simple indicaremos directamente el tipo que puede ser las constantes: TIPO_CARACTER, TIPO_ENTERO, TIPO_DECIMAL, TIPO_FECHA y TIPO_FECHAHORA. Si el valor es un array, el tipo contiene un array asociativo con los tipos de cada columna. Más adelante se puede ver un ejemplo con valor simple, y éste seria un ejemplo con array:

```
$datos = array( array('nombre'=>"L'Eliana", 'superficie'=>'45.6') );
```

```
$conexion->prepararOperacion($datos, array('nombre'=>array('tipo'=>TIPO_CARACTER,), 'superficie'=>array('tipo'=>TIPO_DECIMAL,),));
// en $datos ya tenemos los datos transformados
```

5.1.2.4.4. CalcularSecuenciaBD

Calcula el valor de una secuencia de BD. Consulte en el manual de su SGBD las opciones que ofrece al respecto.

5.1.2.4.5. CalcularSecuencia

Calcula el máximo de una tabla siguiendo varios criterios (según los parámetros).

5.1.2.4.6. preparedQuery

Es similar a consultar y operar, pero usando sentencias preparadas. Es más eficiente que estar formando cada vez la sentencia SQL ya que el SGBD solo tiene que analizar la sentencia la primera vez. No requiere escapar caracteres especiales. Es recomendable especialmente cuando usamos la sentencia dentro de bucles. Al igual que consultar, también tiene un parámetro opcional, que permite pasarle los tipos de los campos. El último parámetro también es opcional y sirve para conservar la sentencia preparada cuando ejecutamos la misma sentencia varias veces con distintos parámetros.

A continuación ponemos un ejemplo de utilización. Comprobamos que, dadas las facturas del año 2006, todas sus líneas esten en estado 'REVISADA', y si no es así actualizamos la factura en concreto pasándola a estado 'SINREVISION' y le añadiremos el campo observaciones que ha introducido el usuario.

```
$conexion = new IgepConexion($dsn);
$sel = "SELECT nfactura as \"nfactura\" FROM lineas WHERE anyo='".2005."' AND estado<>'REVISADA' group by nfactura";
$res = $conexion->consultar($sel);
if ($res!=-1 and count($res[0])>0) {
   //Como no sabemos el contenido de observaciones escapamos por si hay cualquier caracter problematico
   $conexion->prepararOperacion($observaciones, TIPO_CARACTER);
   foreach ($res as $facturaSinRevision){
      $res2 = $conexion->operar("UPDATE factura set observaciones='$observaciones', estado='SINREVISION'
                           WHERE anyo='2005' AND factura='".$facturaSinRevision['nfactura']."'");
      if ($res2==-1) {
         // Mensaje de Error
         ....
      }
}
```

El mismo ejemplo usando sentencias preparadas seria:

```
$conexion = new IgepConexion($dsn);
try {
   $sel = "SELECT nfactura as \"nfactura\" FROM lineas WHERE anyo=? AND estado<>'REVISADA' group by nfactura";
   $res = $conexion->preparedQuery($sel, false, array(2005));
   if (count($res[0])>0) {
      $upd = null;
      $sql = "UPDATE factura set observaciones=?, estado='SINREVISION' WHERE anyo=? AND factura=?";
      //con sentencias preparadas no es necesario escapar caracteres especiales
      foreach ($res as $facturaSinRevision)
         $res2 = $conexion->preparedQuery($sql, true, array($observaciones,'2005',$facturaSinRevision['nfactura']), null, $upd);
   }
} catch (Exception $e) {
   // Mensaje de Error
   ...
}
```

El control de errores con sentencias preparadas siempre es a través de excepciones [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/excepciones.html].

También están disponibles los métodos consultarForUpdate y preparedQueryForUpdate que permiten bloquear los registros que recuperan. Más información aquí [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/trans.html].

5.1.2.5. Utilizar el driver MDB2 o nativo

En algunos casos, puede ser interesante trabajar directamente con el driver PEAR::MDB2 o el driver nativo de PHP. Un ejemplo puede ser la utilización de procedimientos almacenados. Para estos casos el FW ofrece el método getPEARConnection que devuelve el objeto PEAR::MDB2 conectado

```
/*CONEXION ACTIVA DE LA CLASE MANEJADORA*/
//Recogemos el objeto IgepConexion de la conexion activa.
$conPEARMDB2 = $this->getConnection();
//Recogemos el objeto PEAR::MDB2 de la conexion activa.
```

```
$conPEARMDB2 = $this->getConnection()->getPEARConnection();
//Recogemos el driver nativo de PHP de la conexion activa.
$conPHP = $this->getConnection()->getPEARConnection()->getConnection();

/*NUEVA CONEXION*/
//Recogemos el objeto PEAR::MDB2
$this->_conOracle = new IgepConexion($g_dns_ora);
$conPEARMDB2 = $this->_conOracle->getPEARConnection();
//Recogemos el driver nativo de PHP
$conPHP = $this->_conOracle->getPEARConnection()->getConnection();
```

5.1.3. Transacciones

5.1.3.1. Introducción

Actualmente el framework no fija el parámetro autocommit, por lo que estamos usando el valor fijado en los servidores: activado para postgres y mysql, y desactivado para oracle (de hecho, no se puede activar). En próximas versiones se normalizará esta situación. Cuando queramos englobar varias operaciones en una transacción, tendremos que iniciar una transacción o asegurarnos que ya estamos en una. De momento no hay soporte para transacciones anidadas.

Nota

En mysql, para tener soporte transaccional hay que definir las tablas con storages que lo soporten como InnoDB.

El framework inicia una transaccion automaticamente para hacer las operaciones del CRUD. Para el resto de casos, deberemos iniciar ésta de modo explícito. Se puede hacer con el método **empezarTransaccion** de IgepConexion. En las clases de negocio no hace falta hacerlo (ya que el framework inicia la transacción), a menos que queramos que la transacción empiece en preInsertar, preModificar o preInsertar.

Ejemplo:

```
$this->getConnection()->empezarTransaccion();
```

También es habitual empezar una transacción cuando tenemos una conexión sobre un DSN distinto al empleado en el formulario, sobre el que queremos hacer operaciones de actualización.

La transacción acaba con el método **acabarTransaccion**, y recibe un parámetro booleano indicando si ha habido error, y en función de éste se hace un commit o rollback. No hay que llamarlo cuando se trata de la transacción del framework.

5.1.3.2. Control de Concurrencia

Para las operaciones del CRUD, el framework aplica un método simple para evitar que dos usuarios modifiquen simultáneamente un registro que previamente han leído. Consiste en añadir a la condición del where, en los update y delete, todos los campos modificables de la tabla con los valores leídos previamente. Si el registro no se encuentra significa que otro usuario lo ha modificado, y entonces se obliga al usuario a repetir la operación.

Esta aproximación también puede usarse en otros casos, aunque cuando la transacción incluye muchas operaciones, puede ser muy complejo. Para estas situaciones, también se puede usar la opción de bloquear los registros, como se explica a continuación.

5.1.3.2.1. Bloqueos

Las operaciones DML update y delete bloquean implícitamente los registros afectados. Sin embargo hay ocasiones en las que primero leemos información de la BD, y en función de ella actualizamos otra información. Ejemplos de estos casos son al incrementar un saldo, obtener un numero secuencial, ... Para estas situaciones nos interesa bloquear los registros que leemos, y así nos aseguramos que no van a cambiar mientras hacemos la actualización.

En las conexiones con autocommit, hay que tener la precaución de empezar una transacción antes de hacer los bloqueos (o asegurarnos que estamos en una transacción ya empezada). Los bloqueos se liberan cuando la transacción finaliza.

También hay que recordar que al finalizar cada petición al servidor, se cierran las conexiones con las bases de datos, y por tanto también se liberan todos los bloqueos. Eso significa que por ejemplo, no podemos mantener bloqueado un registro mientras el usuario lo edita.

Los bloqueos se hacen sin espera (si el registro está bloqueado por otro usuario se produce un error), con el fin de no colapsar el servidor web. En el caso de mysql no se dispone de la opción de no esperar, aunque la espera se corta tras un timeout. Este timeout por defecto es de 50 segundos, por lo que conviene cambiar el parámetro innodb_lock_wait_timeout a un valor de menos de 5 segundos.

Para hacer estos bloqueos, tenemos en IgepConexion el método **consultarForUpdate** que funciona exactamente como consultar pero bloqueando los registros afectados. Si algún registro ya esta bloqueado por otro usuario, se produce una excepción que habrá que capturar. A continuación tenemos un ejemplo de su utilización:

```
$con->empezarTransaccion();
try {
   $res = $con->consultarForUpdate('select * from tinv_donantes where orden=5');
   if ($res == -1) {
      $this->showMensaje('APL-x'); // algun problema con la consulta
      $con->acabarTransaccion(1);
      return -1;
   }
} catch (gvHidraLockException $e) {
   $this->showMensaje('APL-y'); // algun registro bloqueado
   $con->acabarTransaccion(1);
   return -1;
}
// actualizaciones varias relacionadas con el registro que tenemos bloqueado
//...
$con->acabarTransaccion(0);
```

También podemos hacer lo mismo usando el método **preparedQueryForUpdate**, que funciona similar pero usando sentencias preparadas:

```
$con->empezarTransaccion();
try {
   $res = $con->preparedQueryForUpdate('select * from tinv_donantes where orden=?', array(5,));
} catch (gvHidraLockException $e) {
   $this->showMensaje('APL-x'); // algun registro bloqueado
   $con->acabarTransaccion(1);
   return -1;
} catch (gvHidraSQLException $e) {
   $this->showMensaje('APL-y'); // algun problema con la consulta; con $e->getSqlerror() obtenemos error PEAR
   $con->acabarTransaccion(1);
   return -1;
}
// actualizaciones varias relacionadas con el registro que tenemos bloqueado
//...
$con->acabarTransaccion(0);
```

En el capítulo 7 [135] se explican con más detalle las excepciones que se pueden utilizar.

En la consulta usada para bloquear conviene limitar al máximo el numero de registros obtenidos a los estrictamente necesarios, y no emplear joins ya que se bloquearían los registros afectados de todas las tablas. De esta manera aumentamos el nivel de concurrencia y reducimos la posibilidad de error por registros bloqueados.

5.1.4. Procedimientos almacenados

En este apartado intentaremos explicar como utilizar procedimientos almacenados en gvHIDRA

5.1.4.1. Introducción

Puede que nuestro sistema de información se haya diseñado depositando toda la lógica de la aplicación en procedimientos almacenados o que, puntualmente, tengamos la necesidad de acceder a uno de esos recursos del SGBD. Para ellos, en este apartado daremos algunos ejemplos de como hacerlo a través de las conexiones del FW.

5.1.4.2. Creación de una conexión

Las conexiones al framework están organizadas en tres capas (capa DBMS-gvHIDRA, capa MDB2 y driver nativo PHP). Las llamadas a procedimientos almacenados se pueden realizar desde cualquier a de las capas, pero ganamos en versatilidad a medida que nos acercamos al driver nativo.

La capa DBMS-gvHIDRA actualmente sólo te permite trabajar mediante el método operar, lo cual, excepto para casos muy puntuales, no parece cómodo. La capa MDB2 tiene métodos como el **executeStoredProc** especialmente para estos casos. De todos modos, nuestra recomendación es utilizar el driver nativo ya que parece que en los dos casos anteriores no se trabaja bien con valores entrada/salida.

```
$conexion = new IgepConexion($dsn);
$con = $conexion->getPEARConnection()->getConnection();
$query="begin proced('1234', :retorno, :msgproc); end;";
$id_result= @OCIParse( $con, $query);
if (!id_result or OCIError($id_result))
       return;
OCIBindByName ( $id_result, ":retorno", $ret, 10 );
OCIBindByName ( $id_result, ":msgproc", $msg, 200 );
@OCIExecute ( $id_result, OCI_DEFAULT );//ATENCION A partir de PHP 5.3.2 cambiar OCI_DEFAULT por OCI_NO_AUTO_COMMIT
```

5.1.5. Recomendaciones en el uso de SQL

En este documento se dan algunas recomendaciones para que las sentencias SQL usadas en gvHidra sean lo más estándar posible, y minimizar el impacto de un posible cambio de gestor de base de datos (SGBD). Los SGBD considerados son PostgreSQL, Oracle y Mysql.

5.1.5.1. Alias en las columnas

Se recomienda poner siempre alias en los campos de una consulta.

Si no usamos los alias, en oracle los nombres de las columnas siempre saldrán en mayúsculas, con lo que no funcionará si cambiamos o otro SGBD. En PEAR::MDB2 hay una opción de compatibilidad que transforma todo a minúscula, pero como no permite tener identificadores mixtos (del tipo 'NombreProveedor'), se ha optado por no habilitarla. Para que el alias tenga efecto hay que ponerlo con comillas dobles:

```
select dpro as "nombre"
from provincias
```

5.1.5.2. Comprobación de nulos

No usar NVL ni DECODE en oracle ya que no existen en Postgresql ni en Mysql. Mejor usar el case que funciona en todos y es estándar SQL (el coalesce es más parecido al nvl y también es estándar, aunque no funciona con oracle 8).

```
SELECT CASE WHEN cpro is null THEN '-' ELSE cpro END
FROM tcom_usuarios
```

5.1.5.3. Concatenaciones

Las concatenaciones se hacen con la función 'concat(str1,str2)' (estándar, en Mysql y oracle ya existe, en postgresql se crea (con el script a continuación); el operador '||' funcionaba en oracle y postgres aunque no se recomienda).

```
CREATE OR REPLACE FUNCTION concat(text, text)
  RETURNS text AS
$BODY$
  BEGIN
    RETURN coalesce($1,'') || coalesce($2,'');
  END;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;

GRANT EXECUTE ON FUNCTION concat(text, text) TO public;
```

5.2. Web Services

- Web Services en PHP [117]
  - Cliente [117]

5.2.1. Web Services en PHP

5.2.1.1. Cliente

La creación de un cliente de un servicio web con PHP es relativamente sencilla haciendo uso de PHP-SOAP. Con la descripción del servicio al que queremos acceder (fichero wsdl), obtendremos acceso a todos los métodos que ofrece el servicio web. A continuación mostramos un ejemplo donde se verá más claramente lo expuesto. Concretamente en el ejemplo llamamos a un WS que, dada una cadena, devuelve la cadena al revés.

```
$objClienteWS = new SoapClient('Ws_Ejemplo.wsdl');
$resultado = $objClienteWS->ejemplo('Hola');
print_r($resultado);
```

De la ejecución de este cliente obtenemos el siguiente resultado:

```
  aloH
```

5.2.1.2. Servidor

La creación del servidor requiere, evidentemente, algo más de trabajo que la del cliente. En este punto haremos un pequeño resumen de los pasos a seguir. Primero tenemos que crear un fichero php (en nuestro ejemplo server.php) que contendrá las llamadas a las clases SOAP correspondientes al servidor. En este mismo fichero se puede incluir la definición de la clase que implementará todos los métodos exportados. Siguiendo con nuestro ejemplo, tenemos que tener un método que nos devuelva la inversa de una cadena. El contenido del fichero es:

```
require_once 'SOAP/Server.php';

 class Prueba_Server {
    function ejemplo($cadena){
      return strrev($cadena);
    }
 }

 $server = new SOAP_Server;
 $server->_auto_translation = true;
 $soapclass = new Prueba_Server();
 $server->addObjectMap($soapclass,'urn:Prueba_Server');
 $server->service($HTTP_RAW_POST_DATA);
```

Para que el cliente tenga acceso a la información que ofrece el WS, necesita de la definición de los métodos exportados por la clase. Esto se obtiene a partir del fichero WSDL. El fichero de nuestro ejemplo es el siguiente:

```
<?xml version="1.0"?>
<definitions name="ServerExample" targetNamespace="urn:ServerExample"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="urn:ServerExample"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

<types xmlns="http://schemas.xmlsoap.org/wsdl/"></types>
<message name="ejemploRequest">
  <part name="cadena" type="xsd:string" />
</message>
<message name="ejemploResponse">
  <part name="cadena" type="xsd:string" />
</message>
```

```
<portType name="ServerExamplePort">
  <operation name="ejemplo">
    <input message="tns:ejemploRequest" />
    <output message="tns:ejemploResponse" />
  </operation>
</portType>
<binding name="ServerExampleBinding" type="tns:ServerExamplePort">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="ejemplo">
    <soap:operation soapAction="urn:Prueba_Server#prueba_server#ejemplo" />
     <input>
       <soap:body use="encoded" namespace="urn:Prueba_Server" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
     </input>
     <output>
       <soap:body use="encoded" namespace="urn:Prueba_Server" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
     </output>
  </operation>
</binding>
<service name="ServerExampleService">
 <documentation />
  <port name="ServerExamplePort" binding="tns:ServerExampleBinding">
   <soap:address location="http://mi-servidor.es/cin/ws/server/server.php" />
  </port>
</service>
</definitions>
```

Con esto nuestro Web Service ya esta funcionando. Simplemente tenemos que llamarlo desde el cliente.

5.2.2. Generación del WSDL

Uno de los puntos más costosos de la creación de un WS en PHP es la generación del WSDL. Hay una serie de herramientas que te permiten su generación siempre y cuando rellenes ciertos parámetros. El propio PEAR::SOAP te facilita un proceso para hacerlo, y es el que vamos a intentar explicar a continuación.

La idea consiste en crear un constructor para nuestra clase y, en él, sobrescribir una serie de arrays donde aportaremos la información de los métodos que exporta una clase. Concretamente se trata de la estructura $this->__dispatch_map que tendrá una entrada por cada uno de los métodos soportados. Después, tras las llamadas que realizamos al SOAP_Server tenemos que utilizar una clase de SOAP que nos generará el WSDL.

El código para nuestro ejemplo es el siguiente:

```
require_once 'SOAP/Server.php';

class Prueba_Server {
   function Prueba_Server(){
       $this->__dispatch_map['ejemplo'] =
           array(
               'in' => array('cadena' => 'string',
                         ),
               'out' => array('cadena' => 'string'),
           );
   }
   function ejemplo($cadena){
     return strrev($cadena);
   }
}


$server = new SOAP_Server;
$server->_auto_translation = true;
$soapclass = new Prueba_Server();
$server->addObjectMap($soapclass,'urn:Prueba_Server');

if(isset($_REQUEST['wsdl'])){
  require_once 'SOAP/Disco.php';
  $disco = new SOAP_DISCO_Server($server,'ServerExample');
  header("Content-type: text/xml");
  echo $disco->getWSDL();
  return;
}

$server->service($HTTP_RAW_POST_DATA);
```

Con este código, al interrogar al server.php directamente, obtendremos un xml que contiene la definición del WS. Este contenido se almacena en un fichero WSDL y, de ese modo, los clientes podrán acceder al servicio.

5.2.3. Web Services en gvHIDRA

Ya hemos visto como se implementan los WS en PHP. Para aplicaciones con gvHidra, se han implementado algunas clases que ayudan a la creación tanto del servidor como del cliente en una aplicación. De momento se utiliza un procedimiento muy básico para controlar la seguridad.

5.2.3.1. Cliente

La creación del cliente se realiza mediante la clase IgepWS_Client. A continuación mostramos el código:

```
$objCIN = IgepWS_Client::getClient('actions/ws/WSCIN.wsdl');
$credencial = IgepWS_Client::getCredential('WSCIN');
$objCIN->getImporte($credencial,....
```

El parámetro credencial contiene los parámetros de validación válidos para ese cliente WS. Este parámetro se obtiene consultando los DSNs de la aplicación. La estructura de la credencial seria la siguiente:

```
$credencial = array(
              'login'=>'XXXXX',
              'password'=>'XXXXX'
   );
```

Esta clase usa SoapClient, y si necesitamos pasarle otras opciones podemos hacerlo con el parámetro opcional. En el ejemplo siguiente se modifican las opciones para poder depurar la comunicación con el servidor. En la web de PHP [http://www.php.net/manual/en/soapclient.construct.php] se pueden ver otras opciones posibles.

```
$obj = IgepWS_Client::getClient('actions/ws/WSCIN.wsdl', array("exceptions" => 0, 'trace'=>1, ));
print "<pre>\n";
print "Request :       \n". htmlspecialchars($obj->__getLastRequest())  ."\n";
print "RequestHeaders :\n". $obj->__getLastRequestHeaders()            ."\n";
print "Response:       \n". htmlspecialchars($obj->__getLastResponse()) ."\n";
print "ResponseHeaders:\n". $obj->__getLastResponseHeaders()           ."\n";
print "Functions:      \n". var_export($obj->__getFunctions(),true)     ."\n";
print "Types:          \n". var_export($obj->__getTypes(),true)         ."\n";
print "</pre>";
```

Para depuración, también puede ser útil inhabilitar la cache del cliente, ya que así podemos ir modificando el wsdl:

```
IgepWS_Client::disableCache();
```

Si el cliente usa una codificación distinta a UTF-8 (como ocurre con gvHIDRA que usa latin1), conviene indicarlo en las opciones de conexión y así no hay que hacer conversiones explicitas, sino que PHP convierte los parámetros de entrada desde la codificación origen a UTF-8, y el resultado lo convierte de UTF-8 a la codificación indicada. Si no lo indicamos se fija a 'latin1'.

Ejemplo:

```
$obj = IgepWS_Client::getClient('actions/ws/WSCIN.wsdl', array("exceptions" => 0, 'trace'=>1, ));
...
print "Response:\n".htmlspecialchars(utf8_decode($obj->__getLastResponse())."\n";
...
```

5.2.3.2. Servidor

En el caso del servidor, los beneficios son mayores. Tenemos dos clases, una clase estática que genera el código básico del Servidor y otra que proporciona un comportamiento al servidor propio de una aplicación gvHidra. La primera de ellas es la clase estática IgepWS_ServerBase. Esta clase simplifica la creación y el registro de un servidor SOAP de WS. Concretamente, en nuestro fichero de lanzamiento del servidor (típicamente server.php), que tiene que estar en el raiz de la aplicación, tendríamos el siguiente código:

```
 require_once "igep/include/igep_ws/IgepWS_ServerBase.php";
 require_once 'ws/server/WSCIN.php';

 IgepWS_ServerBase::registrar('WSCIN');
```

Por otro lado, tenemos que crear la clase que controla el Servidor (en nuestro ejemplo WSCIN). Esta clase, al heredar de IgepWS_Server tiene el mecanismo de validación ya implementado y el sistema de conexión propio de gvHidra. La única premisa que se exige es que, si se requiere validación, el método implementado por el programador debe incluir un parámetro $credencial que se pasará al método checkCredential para validar su contenido.

A continuación mostramos un ejemplo:

```
include_once "igep/include/igep_ws/IgepWS_Server.php";
```

```
class WSCIN extends IgepWS_Server
{
    function __construct()
    {
        $msgs = array(
                '1'=>'Error de conexión. Avise al Servicio de Informática',
                '2'=>'Error en operación. Avise al Servicio de Informática',
        );
        parent::IgepWS_Server('WSCIN', $msgs);
        ...
    }

    function getImporte($credencial,$anyox, $dgralx, $numx, $tipo_expx, $numtipo_expx)
    {
      if (!$this->checkCredential($credencial))
        return $this->getAuthError();
      $dsn = ConfigFramework::getConfig()->getDSN('dsn_cin');
      $db = $this->conectar($dsn);
      if (!$db)
        return $this->gvhFault('1');
      ...
      return array('implic' => floatval($res[0]['implic']), 'impadj' => floatval($res[0]['impadj']));
    }
    ...
}
```

Si nuestro servidor acepta varias credenciales, podemos pasarle un vector al constructor del padre:

```
parent::IgepWS_Server(array('WSCIN','WSMCMENOR',), $msgs);
```

En las credenciales de los servidores de web services, la contraseña hay que almacenarla con hash. Para ello usar el formulario en igep/include/igep_utils/protectdata.php para obtener los hash de las contraseñas, y guardar estas últimas en un lugar seguro, fuera de la aplicación.

El parámetro con los mensajes de error se utiliza si provocamos los errores (Soap_Fault) con el método gvhFault, que hace las conversiones necesarias en la codificación y tiene un parámetro opcional usado para que nos informe del error en el debug (tener en cuenta que los errores de conexión y los del método consultar ya se registran en el debug). Por ejemplo:

```
return $this->gvhFault('2','Error consultando tabla');
```

Si queremos restringir algún método a algunas credenciales, podemos hacerlo con método checkCredential pasándole la lista de credenciales:

```
if (!$this->checkCredential($credencial, array('WSMCMENOR',)))
        return $this->getAuthError();
```

También hay que tener precaución cuando hacemos consultas sobre base de datos, que si utilizamos campos calculados o agregados (count, min, ...), el resultado será de tipo string. Si queremos obtener otro tipo tendremos que modificarlo usando la función floatval para tipo float, intval para tipo int, ... En caso de problemas podemos forzar las conversiones con usando la clase SOAP_Value donde básicamente le indicamos el nombre del atributo, el tipo y el valor.

Ejemplo:

```
$modulos_usuario = new SOAP_Value('modulos', 'ModulosValidaStruct', $array_con_modulos);
```

También es importante tener en cuenta la codificación, sobretodo cuando nuestro WS recibe o devuelve campos string que puedan tener carácteres especiales.

Cuestiones:

- la codificación usada en los WS es UTF-8, luego habrá que hacer las transformaciones necesarias desde/hacia LATIN-1 (la codificación usada en gvHIDRA)
- si queremos retornar una cadena obtenida de la BD o de una constante en un fichero fuente de PHP, tenemos que transformarla a utf-8 con utf8_encode($cadena), o mejor usamos el método encode.
- si recibimos un parámetro texto vendrá en utf-8, luego también habrá que transformarlo (utf8_decode) a latin-1 para operar con él (concatenar con otras cadenas, almacenar en BD, ...). Lo podemos hacer con el método decode.

- en caso de problemas también podemos hacer las transformaciones con iconv [http://www.php.net/iconv].

En caso de problemas con algún tipo de datos conviene consultar la documentación en http://www.w3.org/TR/xmlschema-2.

5.3. ORM

En este apartado vamos a hablar de como integrar un ORM en el framework gvHIDRA. Actualmente, no se ha desarrollado una clase específica para uso de ORMs, pero es, como veremos en este apartado, sencilla su integración.

5.3.1. Introducción

Entendemos un ORM (mapeo objeto-relacional) como un sistema o técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Utilizar un ORM tiene una serie de ventajas que nos facilitan enormemente tareas comunes y de mantenimiento:

- **Reutilización**: La principal ventaja que aporta un ORM es la reutilización permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.
- **Encapsulación**: La capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.
- **Portabilidad**: Utilizar una capa de abstracción nos permite cambiar en mitad de un proyecto de una base de datos MySQL a una Oracle sin ningún tipo de complicación. Esto es debido a que no utilizamos una sintaxis MySQL, Oracle o SQLite para acceder a nuestro modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.
- **Seguridad**: Los ORM suelen implementar mecanismos de seguridad que protegen nuestra aplicación de los ataques más comunes como SQL Injections.
- **Mantenimiento del código**: Gracias a la correcta ordenación de la capa de datos, modificar y mantener nuestro código es una tarea sencilla.

5.3.2. Ejemplo: Propel ORM

Actualmente existen muchas soluciones ORM con PHP en el mercado, tanto libres como propietario, aunque para este ejemplo hemos optado por un enfoque Open-Source con el objetivo de seguir con la misma dirección que gvHidra. Como se ha indicado anteriormente, cualquier otra solución sería válida.

Propel es un ORM basado en PHP. Mediante el mapeo de una BBDD en XML se generan un conjunto de clases para su manejo, simplificando de este modo el acceso a los datos, dando una solución CRUD.

5.3.2.1. Características

Según indica la propia documentación oficial de Propel, funciona con los siguientes SGBD's:

- PostgreSQL

- MySQL
- Oracle
- Sql Server
- Sqlite

El trabajo con las clases generadas es cómodo e intuitivo siempre que se tengan nociones de desarrollo con objetos.En un principio aunque la curva de aprendizaje puede parecer mas compleja de lo esperado, por las pruebas realizadas el resultado mediante este ORM es rápido y eficaz. Realmente no cuesta un tiempo excesivo hacer una consulta o inserción en las tablas de la BBDD. Las opciones que nos brinda este ORM son muchas y variadas; van desde realizar consultas sin necesidad de utilizar SQL, hasta controlar la consulta que se va a realizar contra la BD introduciendo una sentencia SQL mas compleja.

El tratamiento de los datos devueltos en las consultas también es un punto importante, ya que se pueden recupera un solo elemento de una consulta, o en caso de necesitarlo, un array de objetos que contienen esa información. Eso significa que de una sola consulta, el ORM nos permite recuperar un conjunto finito de resultados, un conjunto finito de objetos de una clase.

Propel nos da una enorme variedad de opciones, estas nos cubrirán en la mayoría de los casos aquellas funciones o necesidades que tengamos. Gracias a la herencia, si alguno de los casos no estuviera cubierto como debería, entonces tenemos la opción de crearnos nuestra propia clase de gestión de los datos, con las mejoras que consideremos oportunas. Recordemos que disponemos de todas las ventajas de la programación orientada a objetos, como la sobrecarga o la herencia.

También nos da una gran variedad de operaciones, aparte de las básicas de CRUD. A continuación se detallan algunas de las más importantes:

1. *Inserción de filas relacionadas*
2. *Guardado en cascada*
3. *Uso de las relaciones en una consulta*
    - Búsqueda de registros relacionados con otro
    - Consultas embebidas
    - Relaciones uno a uno
    - Relaciones muchos a muchos
    - Reducir al mínimo las consultas
4. *Validaciones*
    - Igual que
    - No igual que
    - Tamaño máximo
    - Tamaño mínimo
    - Tipo
    - Valor válido

- Personalizadas

5. *Transacciones*: Las transacciones de bases de datos son la clave para asegurar la integridad de los datos y el rendimiento de las consultas de base de datos.

6. *Herencia*

   - Igual que

   - No igual que

5.3.2.2. Configuración

La instalación se puede hacer mediante **PEAR**, es importante tener en cuenta que tendremos que haber instalado previamente también Phing, una vez hecho esto, se accede a los canales correspondientes y se instalan los paquetes de Propel.

La fase previa al uso de Propel en la cual se definen las estructuras de datos y las clases manejadoras, se puede abordar de dos modos:

5.3.2.2.1. Desde un XML

Se generan a través de funciones nativas de Propel y mas concretamente "*propel-gen*", una serie de clases manejadoras de la BBDD, tanto las clases base, como unas clases que heredan de estas primeras

Tendremos entonces que generar el SQL para poder crear las tablas en la BBDD, este proceso también es automatizado mediante la herramienta "propel-gen", que nos generará un fichero SQL conteniendo toda la estructura de la BBDD, este fichero lo usaremos para crear en la BBDD las tablas correspondientes.

5.3.2.2.2. Mediante Ingeniería InversaL

Obtendremos el XML con toda la estructura de las tablas implicadas.

Realizamos entonces los mismos pasos que en el primer caso, para poder generar la estructura de las clases manejadoras.

5.3.2.3. Pues en marcha

La instalación se puede hacer mediante **PEAR**, es importante tener en cuenta que tendremos que haber instalado previamente también Phing, una vez hecho esto, se accede a los canales correspondientes y se instalan los paquetes de propel.

Pasos para realizar esta instalación:

1. Se debe agregar el canal pear.propelorm.org al entorno de PEAR.

2. Una vez que tenemos el canal, puede instalar el paquete del generador, o el paquete de tiempo de ejecución, o ambas cosas, si es una primera instalación, preferentemente instalar ambas cosas.

3. Es importante hacer uso de la opción '-a' para permitir que PEAR pueda descargar e instalar las dependencias.

```
pear channel-discover pear.propelorm.org
pear install -a propel/propel_generator
pear install -a propel/propel_runtime
```

El generador de Propel utiliza Phing 2.3.3, si no esta instalado, se debe instalar.

Pasos para realizar esta instalación:

1. Se debe agregar el canal *pear.phing.org* al entorno de PEAR.

2. Posteriormente instalamos el *Phing* y el *Log*.

```
pear channel-discover pear.phing.info
pear install phing/phing
pear install Log
```

Una vez realizadas las instalaciones, simplemente tendremos que hacer uso de “propel-gen” para comprobar que todo ha funcionado correctamente, aunque el proceso no realice ninguna función, ya que no habremos creado los ficheros necesarios para ello.

Podemos recuperar la información, los datos, de la base de datos y prepararla en un fichero XML, para ello Propel tiene herramientas que nos lo permitirán, generándonos un fichero XML que contiene todos estos datos.

Posteriormente se explicaran los pasos previos a seguir para hacer uso de Propel. Antes de poder comenzar a usar Propel, tendremos que preparar la capa de comunicación con nuestra BBDD para lo cual nos vamos a encontrar con dos situaciones:

CASO 1: Partimos de una BBDD en formato XML.

1. Se generan a través de funciones nativas de Propel y mas concretamente “propel-gen”, una serie de clases manejadoras de la BBDD, tanto las clases base, como unas clases que heredan de estas primeras.

2. Tendremos entonces que generar el SQL para poder crear las tablas en la BBDD, este proceso también es automatizado mediante la herramienta “propel-gen”, que nos generará un fichero SQL conteniendo toda la estructura de la BBDD, este fichero lo usaremos para crear en la BBDD las tablas correspondientes.

CASO 2: Partimos de una BBDD ya creada, usamos ingeniería inversa.

1. Obtendremos el XML con toda la estructura de las tablas implicadas.

2. a.Realizamos entonces los mismos pasos que en el primer caso, para poder generar la estructura de las clases manejadoras.

Se detalla a nivel técnico cada uno de los dos casos, con su comportamiento correspondiente.

CASO 1: BBDD en formato XML

1. Crearemos un nuevo fichero llamado *schema.xml* que contendrá la estructura de nuestra BBDD siguiendo el siguiente esqueleto:

```
<?xml version="1.0" encoding="UTF-8"?>
 <database name="" defaultIdMethod="native">
   <table name="" phpName="">
      <!-- column and foreign key definitions go here -->
      <column name="" type="integer" required="true" primaryKey="true" autoIncrement="true"/>
       ...
      <foreign-key foreignTable="" phpName="" refPhpName="">
        <reference local="" foreign=""/>
      </foreign-key>
     …
   </table>
 </database>
```

   En este fichero XML vamos a incluir toda la estructura de la BBDD con la que queremos trabajar, las columnas y las claves implicadas. Estos datos serán los que emplearemos para crear las tablas en nuestro SGBD.

2. Para la construcción final del modelo, necesitamos tanteo el recién creado schema.xml como un nuevo fichero build.properties que contendrá la información de la configuración de la BBDD.

```
# Driver de la BBDD
propel.database = pgsql|mysql|sqlite|mssql|oracle

# Nombre del proyecto
```

```
propel.project = <…>
```

Se ha de tener en cuenta que para el correcto funcionamiento, el nombre del proyecto debe ser el mismo que el de la BBDD.

3. Hemos de dejar estos dos ficheros schema.xml y build.properties en el mismo nivel dentro del subdirectorio del proyecto, solo nos queda hacer la llamada a Propel para que nos genere las clases correspondientes, esto es pasándole el parametro “om”, Object Model generador.

```
Propel-gen om
```

Por cada tabla en la BBDD, Propel crea tres clases de PHP:

- Una clase modelo, lo que representa una fila en la base de datos.
- Una clase Peer, ofreciendo constantes estáticas y métodos sobre todo por compatibilidad con versiones anteriores de Propel.
- Una clase Consulta que sirve para trabajar sobre una tabla para poder recuperar y actualizar filas.

4. Propel además también puede generarnos el fichero con el código SQL completo de nuestras tablas, para ello, el parámetro de llamada debe ser “sql”, no es obligatorio para el funcionamiento de Propel pero puede ser interesante para tener el modelo de datos y poder llevarlo a otro SGBD si lo consideramos necesario.

5. Ajustes de Conexión del Runtime. Ahora se debe agregar un archivo de configuración para que las clases de objetos generadas y el runtime de Propel se puedan conectar a la base de datos, y registrar la actividad de Propel. Para ello crearemos un fichero con el nombre runtime-conf.xml que mantendrá el siguiente esqueleto, el ejemplo define una conexión a MySQL:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<config>
  <!—Uncomment this if you have PEAR Log installed
  <log>
    <type>file</type>
    <name>/path/to/propel.log</name>
    <ident>propel-bookstore</ident>
    <level>7</level>
  </log>

  <propel>
    <datasources default=””>
      <datasource id=””>
        <adapter>mysql</adapter>
        <connection>
          <dsn>mysql:host=localhost;dbname=</dsn>
          <user></user>
          <password></password>
        </connection>
      </datasource>
    </datasources>
  </propel>
</config>
```

6. Ahora creamos la configuración para el Runtime mediante la llamada a

```
propel-gen convert-conf.
```

Esto nos genera en el directorio \conf de nuestra aplicación el siguiente fichero

```
<nombre >-conf.php
```

Donde <nombre> es el nombre del proyecto definido en el fichero *build.properties*

CASO 2: BBDD ya creada en un SGBD, mediante ingeniería inversa.

Partimos de un punto mas próximo a la realidad como es la incorporación de un ORM a un entorno ya existente, por lo que evidentemente tenemos también la BBDD creada y funcionando. El proceso es bastante similar, solo difiere algunos puntos.

1. En primer lugar hemos de generar el fichero *build.properties* el cual tendrá el siguiente código dentro

```
propel.project = <nombre>

# The Propel driver to use for generating SQL, etc.
propel.database = <driver>

# This must be a PDO DSN
propel.database.url = <driver>:dbname=<nombre>
propel.database.user = <user>
# propel.database.password =<pwd>
```

2. Una vez tenemos el fichero build.properties se hace la llamada a Propel para que nos genere el fichero XML con la estructura de la BBDD esta vez haremos uso del parámetro “reverse”.

```
propel-gen reverse
```

   Con esto obtendremos un fichero *schema.xml* que contendrá la especificación de la BBDD en XML.

3. Ahora como en el caso 1, nos queda hacer la llamada a Propel para que nos genere las clases correspondientes, esto es pasandole el parámetro “om”, Object Model generador.

```
propel-gen om
```

   Recordemos que por cada tabla en la BBDD, Propel crea tres clases de PHP:

   - una clase modelo.
   - una clase Peer.
   - una clase Consulta.

4. Llegados a este punto, los pasos a seguir son los mismos que las que tenemos en el caso anterior, a partir del punto 1.d.

Llegados a este punto, ya tenemos la estructura de clases para trabajar con Propel en nuestra aplicación, ahora tendremos que incluir la llamada correspondiente.

Mediante esta llamada se incluye la librería de propel, y además inicializamos las clases particulares para que nuestra aplicación haga uso de ella.

```
// Include the main Propel script
require_once 'propel/Propel.php';

// Initialize Propel with the runtime configuration
Propel::init("/<proyecto>/build/conf/bookstore-conf.php");

// Add the generated 'classes' directory to the include path
set_include_path("/<proyecto>//build/classes" . PATH_SEPARATOR . get_include_path());
```

Capítulo 6. Seguridad

6.1. Autenticación de usuarios

6.1.1. Introducción

gvHIDRA proporciona varios mecanismos para autenticar los usuarios, y también permite crear nuevos. Se usa el PEAR::Auth [http://pear.php.net/manual/en/package.authentication.auth.php], por lo que en algunos puntos en conveniente ver su documentación. Para comprender el funcionamiento, conviene distinguir dos partes:

1. formulario para introducir las credenciales y comprobación de su validez
2. carga de la información relativa al usuario / aplicación. Esta información se carga en la sesión, y su estructura puede verse en los ejemplos del método postLogin en igep/include/valida/AuthBasic.php, igep/custom/cit.gva.es/auth/AuthWS.php y en Autenticación imap gva [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/auth_gva.html].

A continuación veremos los mecanismos disponibles y como se tratan estos puntos en cada uno de ellos.

6.1.1.1. comun

Es el método usado por defecto en el tema 'cit.gva.es'. Aquí el formulario es externo, y en el framework solo se realiza la parte 2). Esta parte se implementa en comun/modulos/valida.php. En el framework se detecta por los parámetros en el GET pasados desde el formulario externo.

Al pulsar opción de salir se cierra la ventana del navegador (ya que se asume que estamos en una ventana nueva).

6.1.1.2. gvPontis

Es el método de ejemplo usado en el tema 'gvpontis'. Muestra un formulario genérico para el paso 1. Los datos para la conexión son usuario 'invitado' y contraseña '1'. En el paso 2 se carga la información necesaria de forma estática.

Para hacer uso de este método hay que copiar el fichero igep/include/valida/login.php a la raíz del proyecto (se puede renombrar), y acceder a él con el navegador.

Al pulsar la opción de salir se vuelve al formulario de conexión.

6.1.1.3. wscmn (Web Service)

Este método es similar al de **comun**, pero sí incluye formulario, y la comprobación se hace con el web service en wscmn. La parte 2) se inicializa haciendo uso de la información recuperada por el web service, por tanto no requiere acceso a base de datos.

Para hacer uso de este método hay que copiar el fichero igep/custom/cit.gva.es/auth/login_ws.php a la raíz del proyecto (se puede renombrar), y acceder a él con el navegador.

Al pulsar la opción de salir se vuelve al formulario de conexión.

6.1.2. Elección del método de Autenticación

Cuando accedemos a la aplicación sin indicar el fichero de inicio, normalmente el servidor web ejecuta el fichero index.php., donde se hacen una serie de comprobaciones para detectar el método que se va a usar. Hay dos formas de elección:

- explícita: el tipo comun se activa con los parámetros pasados en el GET, y para los otros métodos hay que acceder por http al fichero de login.
- implícita: si no hay elección implícita, se buscan los ficheros 'login*.php' en la raíz y se accede al primero, en orden alfabético.

6.1.3. Crear un nuevo método de Autenticación

A continuación se explican los pasos para crear el nuevo método. También podemos consultar la implementación de los tipos gvPontis y wscmn, ya que se han creado de esta forma. También hay disponible un ejemplo [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/auth_gva.html] usando un servidor imap.

Excepto el fichero de indice (paso 1), los otros pueden situarse en cualquier ubicación, y normalmente vendrá determinado en función de donde necesitemos usarlos (aplicación, custom o framework).

6.1.3.1. Paso 1. Clase Principal

En primer lugar tenemos que crear el fichero php que contiene la clase a usar con PEAR::Auth. Esta clase ha de heredar de 'igep/include/valida/gvhBaseAuth.php'. Los métodos que debemos implementar son:

- fetchData: recibe usuario y contraseña y comprueba su validez
- autenticate: es el método llamado desde el paso 1, y si se aceptan las credenciales, llamamos al método open pasándole la URL de la clase que definimos en paso 2. Consultar los ejemplos para más detalles.

6.1.3.2. Paso 2. <DESCRIPCION>

Creamos el fichero que invoca al método autenticate anterior. Este fichero siempre ha de ir en la raiz del proyecto, y es el punto de entrada a la aplicación usando el navegador.

6.1.3.3. Paso 3. <DESCRIPCION>

Creamos la clase validacion, con el método valida (llamado desde el framework), que lo que hace es cargar la información necesaria en la sesión. Podemos llamar al método checkData (de la clase principal) que comprueba si en la sesión está toda la información que requiere gvHIDRA.

6.2. Modulos y Roles

6.2.1. Introducción

Los módulos y roles es la forma usada para controlar el acceso a las distintas partes de un aplicación. No hay que confundir estos módulos con los usados a nivel de los menús para agrupar funcionalidades.

6.2.1.1. Módulos

Los módulos representan permisos que un usuario tiene en una aplicación determinada. Los módulos se asignan cuando se lleva a cabo una asociación entre un usuario y una aplicación, y se cargan en el inicio de la aplicación. Cada módulo tiene un código, una descripción y opcionalmente puede tener un valor. Cada usuario puede tener asignado uno o varios módulos, y para cada uno puede modificar su valor. Algunos usos habituales de módulos son el controlar si un usuario puede acceder a una opción de menú, o si puede ver/editar un campo determinado de una pantalla, o para guardar alguna información necesaria para la aplicación (departamento del usuario, año de la contabilidad, ...). También suelen

usarse para definir variables globales para todos los usuarios, aunque en este caso es recomendable asignarlos a roles (ver apartado siguiente).

Ejemplo:

- Todos los usuarios que consulten la aplicación PRESUPUESTARIO deben tener el módulo M_CONSUL_PRESUPUESTARIO, es decir, nadie que no tenga ese módulo asociado podrá acceder al apartado de listados de la aplicación
- Sólo el personal de la oficina presupuestaria tendrá acceso a la manipulación de datos, es decir el módulo M_MANT_PRESUPUESTARIO
- Sólo técnicos y el jefe de la oficina presupuestaria tendrán tendrán acceso a la entrada de menú "control de crédito". Pues serán aquellos usuarios que tengan el módulo M_MANT_PRESUPUESTARIO, con el valor TECNICO o el valor JEFE.
- Usuarios:
  - Sandra (Administrativa de otro departamento que consulta la aplicación): Tiene el módulo M_CONSUL_PRESUPUESTARIO
  - Juan (Administrativo): Tiene el módulo M_MANT_PRESUPUESTARIO, (bien sin valor, o con el valor PERFIL_ADMD)
  - Pepe (Técnico): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO
  - Pilar (Jefa de la oficina presupuestaria): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_JEFE
- Módulos:
  - Nombre: M_CONSUL_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las consultas de la aplicación de presupuestario
    - Valores posibles: <COMPLETAR>
  - Nombre: M_MANT_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las opciones de mantenimiento de la aplicación de presupuestario
    - Valores posibles: PERFIL_ADMD, PERFIL_TECNICO o PERFIL_JEFE

6.2.1.2. Roles

Los roles representan el papel que el usuario desempeña en una aplicación y a diferencia de los módulos, cada usuario sólo puede tener uno y no tienen valor. ¿Entonces para que queremos los roles si podemos hacer lo mismo usando módulos sin valor? Podemos ver los módulos como los permisos básicos, y los roles como agrupaciones de módulos. Realmente los roles se utilizan para facilitar la gestión de usuarios. Si sólo usamos módulos y por ejemplo tuviéramos 30 módulos, seria muy difícil clasificar a los usuarios ya que seguramente cada uno tendría una combinación distinta de módulos, y cada vez que hubiera que dar de alta un usuario tendríamos que tener un criterio claro para saber cuales de los módulos asignar. Con roles es más simple, ya que es sencillo ver los permisos de cualquier usuario (viendo el role suele ser suficiente), y para añadir nuevos usuarios normalmente solo necesitaremos saber su role. Para que esto sea fácil de gestionar, tendríamos que tener algún mecanismo que nos permitiera asignar módulos a roles, y que los usuarios con un role "heredaran" estos módulos. Lo más flexible seria tener esta información en tablas aunque también se podría empezar haciéndolo directamente en PHP (o ver abajo módulos dinámicos):

```
if ($role == 'admin') {
```

```
   // combinacion 1
   $modulos[] = array('borrarTodo'=>array('descrip'=>'borra todo',));
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion11'=>array('descrip'=>'opcion 11',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   $modulos[] = array('opcion13'=>array('descrip'=>'opcion 13',));
   ...
} elseif ($role == 'gestor') {
   // combinacion 2
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   ...
} elseif ($role == '...') {
...
```

De esta forma, añadir un nuevo usuario de tipo administrador consistiría simplemente en indicar su role, en vez de tener que asignarle los N módulos que tienen los administradores.

La solución más flexible seria usar sólo módulos para controlar cualquier característica que pueda ser configurable por usuario, y asignar los módulos a los roles de forma centralizada (bien en base de datos o en el inicio de la aplicación). De esta forma el programador solo tendría que tratar con los módulos, y el analista con los módulos de cada role.

El ejemplo anterior usando roles podría ser:

- módulos: M_CONSUL_PRESUPUESTARIO, M_MANT_PRESUPUESTARIO (ambos sin valor)
- roles:
  - PERFIL_ADMD (módulo M_MANT_PRESUPUESTARIO), asignado a Juan
  - PERFIL_TECNICO (módulo M_MANT_PRESUPUESTARIO), asignado a Pepe
  - PERFIL_JEFE (módulo M_MANT_PRESUPUESTARIO), asignado a Pilar
  - PERFIL_OTROS (módulo M_CONSUL_PRESUPUESTARIO), asignado a Sandra

En este caso las dos soluciones tienen una complejidad similar, aunque las diferencias serán más evidentes conforme aumenten el número de módulos y usuarios. Si por ejemplo decidiéramos que ahora todos los técnicos han de tener un nuevo módulo X, sin usar roles tendríamos que añadir ese módulo a todos los usuarios que tuvieran el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO; usando roles seria simplemente añadir el módulo X al role PERFIL_TECNICO.

6.2.2. Uso en el framework

El primer ejemplo de uso de módulos puede encontrarse en la creación de los ficheros xml que da lugar a la pantalla de inicio de la aplicación [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/menu.html], en dichos ficheros se utiliza la etiqueta "controlAcceso" para indicar que módulos necesita tener asignados un usuario para acceder a la opción. Por ejemplo:

```
<opcion titulo="Prueba del árbol" descripcion="Pues eso... prueba del árbol"
 urlAbs="phrame.php?action=IgepPruebaArbol__iniciarVentana" imagen="menu/24.gif">
   <controlAcceso><moduloPermitido id="M_INTRANET"/></controlAcceso>
</opcion>
```

Con el párrafo anterior de XML, se indica que la entrada de la aplicación "Prueba del árbol" sólo estará disponible para aquellos usuarios que cuenten entre sus módulos el M_INTRANET. También se puede hacer el control usando un role.

Los módulos podemos usarlos en otros sitios, y podemos acceder a ellos a través de

Tabla 6.1. Tabla de metodos 1

método	descripción
IgepSession::hayModulo(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo.

método	descripción
IgepSession::dameModulo(<modulo>)	Información del módulo.
ComunSession::dameModulos()	Todos los módulos (estáticos) asignados al usuario (se recomienda usar el método con el mismo nombre de IgepSession).

Cuando queremos usar módulos que no estén permanentemente asignados a un usuario, sino que la asignación dependa de otras circunstancias que puedan ir cambiando durante la ejecución de la aplicación, la asignación no la haremos en el inicio de la aplicación. Para poder añadir o eliminar módulos en tiempo de ejecución, y conseguir, entre otras cosas el poder hacer accesibles u ocultar opciones de menú dinámicamente, podemos además usar:

Tabla 6.2. Tabla de metodos 2

método	descripción
IgepSession::anyadeModuloValor(<modulo>, <valor>=null, <descripcion>=null)	Añade un nuevo módulo al usuario
IgepSession::quitaModuloValor(<modulo>, <valor>=null)	Quita el módulo al usuario; si se le pasa valor, sólo lo quita si el valor del módulo coincide con el valor pasado
IgepSession::hayModuloDinamico(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo dinámico
IgepSession::dameModuloDinamico(<modulo>)	Información del módulo dinámico
IgepSession::dameModulosDinamicos()	Todos los módulos dinámicos asignados al usuario
IgepSession::dameModulos()	Todos los módulos asignados al usuario

Estos módulos les llamaremos módulos dinámicos. A efectos del framework, no hay diferencia entre los módulos cargados inicialmente y los módulos dinámicos. El plugin CWPantallaEntrada ya incluye dicha funcionalidad, por lo que si en alguno de los ficheros XML [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/menu.html] aparece una opción como esta:

```
<opcion titulo="Prueba de módulo dinamico"
 descripcion="Ejemplo de activación y descativación de opciones en ejecución"
 urlAbs="http://www.gvpontis.gva.es" imagen="menu/24.gif">
   <controlAcceso><moduloPermitido id="MD_GVA"/></controlAcceso>
</opcion>
```

hasta que en algún punto de nuestro código PHP no realicemos una llamada como esta:

```
IgepSession::anyadeModuloValor("MD_GVA")
```

la opción permanecerá oculta.

Si después de habilitarla queremos volver a ocultarla, basta con ejecutar la opción:

```
IgepSession::quitaModuloValor("MD_GVA")
```

Podemos obtener el role del usuario con el método IgepSession::dameRol().

6.2.2.1. Restricciones en los menús

Para que los cambios sean efectivos de forma visual en los menús, es necesario que se reinterprete el código XML, cosa que sólo se hace cuando se pasa por la pantalla principal de la aplicación, mientras tanto NO será actualizada la visibilidad/invisibilidad de las distintas ramas, módulos y/o opciones. Por tanto, convendrá también añadir en las acciones que correspondan al menú, comprobación explícita de que se tienen los módulos/roles necesarios para el acceso.

6.3. Permisos

<PENDIENTE>

Parte IV. Conceptos Avanzados

Tabla de contenidos

Capítulo 7. Conceptos Avanzados

7.1. Excepciones

Con el objetivo de hacer la gestión de errores más flexible para el programador, existen algunas funcionalidades de gvHIDRA que usan excepciones. Para ello se ha creado una jerarquia de excepciones que irá creciendo según las necesidades. Esta jerarquía de excepciones es la siguiente:

Tabla 7.1. Tabla de Excepciones

excepción	descripción
Exception	Excepción definida en PHP
gvHidraException	Excepción usada como base de todas las excepciones del framework
gvHidraSQLException	cCase base para excepciones relacionadas con SQL
gvHidraLockException	Excepción producida cuando no se puede bloquear un recurso. Ver bloqueos [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/trans.html].
gvHidraPrepareException	En sentencias preparadas [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/conexionesAlternativas.htm], cuando no se puede preparar una sentencia
gvHidraExecuteException	En sentencias preparadas [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/conexionesAlternativas.htm], cuando no se puede ejecutar una sentencia
gvHidraFetchException	En sentencias preparadas [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/conexionesAlternativas.htm], cuando no se puede recuperar datos
gvHidraNotInTransException	Excepción producida cuando en una operación se requiere transacción en curso.

A continuación vamos a explicar algunos métodos disponibles.

7.1.1. gvHidraSQLException

Define una propiedad para almacenar el objeto error del PEAR. El objeto se asigna en el constructor, y se puede recuperar con un método.

Métodos:

- **__construct($message='', $code=0, $prev_excep=null, $pear_err=null)**

  Posibilidad de asignar objeto error. El tercer parámetro sólo tiene efecto a partir de PHP 5.3.

- **getSqlerror()**

  Obtener el objeto error.

7.1.2. gvHidraLockException

7.1.3. gvHidraPrepareException

7.1.4. gvHidraExecuteException

7.1.5. gvHidraFetchException

7.1.6. gvHidraNotInTransException

7.2. Log de Eventos

Funcionamiento del log y los diferentes usos que se le puede dar.

Ponemos especial énfasis en su utilización como método de Debug del código en desarrollo.

7.2.1. Introducción

El objetivo de este módulo es registrar ciertos eventos, acciones, movimientos, operaciones, ... para poder averiguar lo que está haciendo la aplicación. Por defecto éstos se registran en una tabla de sistema. Esta tabla se llama tcmn_errlog y hay que crearla [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/script-tabla-log.html] previamente a usarla. Más adelante se pueden definir otros métodos como podría ser el correo, ficheros, ... o mejor Pear::Log [http://pear.php.net/package/Log]. En los ficheros de configuración [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/configphp.html] (propiedad DSNZone/dbDSN con id 'gvh_dsn_log') se puede definir una fuente de datos válida para añadir registros a la tabla.

Esto habitualmente lo usaremos para detectar errores de una aplicación en explotación o para poder realizar una traza detallada de una acción en desarrollo.

Hemos realizado una clasificación de los eventos que se registran en la tabla dependiendo de su gravedad:

Tabla 7.2. Tabla de clasificación de los eventos

tipo	descripción
PANIC	Errores graves, la aplicación no deberia seguir ejecutándose
ERROR	Errores
WARNING	Errores menores
NOTICE	Información a nivel de Auditoría
DEBUG_USER	Mensajes de traza (debug) del programador
DEBUG_IGEP	Mensajes de traza (debug) de gvHidra

Por defecto se registran todos los eventos de los dos primeros tipos. Si queremos cambiarlo, podemos hacerlo indicando a la aplicación el nivel de sensibilidad. Es decir, indicar que tipo de eventos se tienen que registrar. Para ello se puede escoger de los siguientes valores:

- **LOG_NONE** : No registra nada.
- **LOG_ERRORS**: Registra únicamente ERROR y PANIC. (valor por defecto)
- **LOG_AUDIT**: Registra todos los anteriores más WARNING y NOTICE.
- **LOG_ALL**: Registra todas las acciones.

El valor lo podemos poner de forma estática en el atributo logSettings del fichero gvHidraConfig.inc.xml de la aplicación o de forma dinámica con el método ConfigFramework->setLogStatus (ver configuración).

Hay que tener precaución con esta variable, para que en explotación no se genere más información que la necesaria. Normalmente le asignaremos un valor u otro en función de si estamos en producción o no.

7.2.2. Crear eventos en el log

Un programador puede crear eventos en el log simplemente llamando al método setDebug de la clase IgepDebug.

Ejemplo:

```
function creacionInformes(){
  IgepDebug::setDebug(DEBUG_USER,'Pasamos a crear los informes');
  ....
}
```

En este ejemplo, y dependiendo del valor del atributo logSettings, el programador inserta una entrada en el log que indica que se ha pasado por el método creacionInformes.

El programador podrá hacer uso del log indicando la gravedad del mensaje que desee atendiendo a la clasificación. Se recomienda hacer uso de DEBUG_USER si se esta realizando un debug en desarrollo y WARNING o NOTICE si se quiere realizar auditoría de lo que realicen los usuarios en explotación.

7.2.3. Consulta del Log

Para ver los eventos generados, se ha creado una consola que permite consultar lo que se inserta en la tabla de sistema. Se recomienda a los programadores que hagan uso de ella añadiendo en el fichero menuHerramientas.xml la siguiente linea:

```
<opcion titulo="Log Aplicación" descripcion="Log Aplicación" urlAbs="igep/_debugger.php" imagen="menu/51.gif"/>
```

Hay que tener precaución en este último punto para que esta opción no esté accesible para los usuarios finales.

7.3. Depurando mi aplicación

<EJEMPLO PRACTICO DE COMO DEPURAR UNA APP>

7.4. Envio de correo desde mi aplicación

Conjunto de métodos para el envio de correos en gvHidra

Esta es una clase que hace uso del paquete PEAR::Mail para el envio de correos. Por defecto no está incluida, por lo que para usarla hay que hacerlo previamente, por ejemplo en include/include.php de la aplicación.

```
include_once "igep/include/IgepCorreo.php";
```

Es una clase estática, luego no hace falta crear instancia para ejectar sus métodos.

7.4.1. Métodos básicos

Sobre estos se basan el resto. Son los que envian correo con o sin anexos. Cuando estamos en desarrollo, le pone como destinatario el correo del usuario que se ha conectado, y la lista de destinatarios original se añade al final del cuerpo del mensaje. De esta forma podemos probar sin preocuparnos de estar enviando correos de prueba a los usuarios.

Cuando hay una lista de destinatarios, el mensaje se envia individualmente a cada uno, por lo que cada destinatario no conocerá el resto de destinatarios. Devuelve falso si falla el envio a alguno de los destinatarios, o cierto en otro caso.

7.4.1.1. sinAnexo

Parámetros:

- $from
- $to: array con los destinatarios del correo
- $subject: asunto
- $msg: cuerpo del mensaje
- $responder_a
- $poner_dest=FALSE: es opcional, y si es true añade al principio del cuerpo la lista de direcciones a las que se envia el mensaje.

7.4.1.2. conAnexo

Parámetros (además de los del método sinAnexo):

- $tmp_fich
- $tipo_fich
- $nom_fich

Estos parámetros son los necesarios para enviar un fichero como anexo. El método sólo está preparado para enviar un anexo.

7.4.2. Otros métodos

La mayor parte de estos métodos hacen uso de las tablas comunes de usuarios, módulos, ... por lo que son poco útiles fuera de la CIT.

7.4.2.1. correoUsuario

Recibe como entrada la cuenta del usuario en la base de datos, y devuelve su dirección de correo en un array.

7.4.2.2. correoNREGPGV

Recibe como entrada un número de registro de personal o un array de ellos y devuelve la lista con sus direcciones de correo.

7.4.2.3. correoListaUsuariosModulo

A partir de una aplicación y un módulo, devuelve las direcciones de correo de los usuarios que lo tienen asignado.

7.4.2.4. correoListaUsuariosAplicacion

Recibe como parámetro la aplicación y el tipo y en reglas generales devuelve las direcciones de los usuarios de la aplicación. En función del tipo hay algunos matices:

1. Se excluyen los usuarios de tipo role, y sólo aquellos usuarios de tipo N (normal) o S (sólo listas)
2. ...

Si se necesitan nuevas combinaciones pedirlas.

7.4.2.5. correoLista

Interno, recibe una consulta y forma un array con el resultado. El campo con el correo ha de llamarse 'dircorreo'.

Ejemplo:

```
IgepCorreo::correoLista("select u.dcorreoint as \"dircorreo\" from tcom_usuarios u where u.nrp in ('".$str_nregpgv."')");
```

7.5. Creación de un custom propio para una aplicación de gvHIDRA

El propósito del presente manual es servir como una guía para que, partiendo de un custom de aplicación básico (el custom "default" incluido en la distribución), podamos modificar/añadir características propias para nuestra aplicación .

El manual no explica sintaxis u opciones del lenguaje de hojas de estilo css, para ello ya hay varios tutoriales en la web que se pueden consultar par un conocimiento mas amplio de css.

Lo que se especificará a lo largo del manual será, la correspondencia entre las diferentes partes de una ventana de una aplicación gvHIDRA (barra superior, inferior, paneles, botones, etc), y los clases o secciones dentro del archivo .css responsables de controlar las características y apariencias de dichas partes .

7.5.1. Pasos previos

Para no empezar desde cero y ahorrarnos tiempo de teclear muchas directivas, primero copiamos un custom base que esta incluido en la distribución (el directorio "default" que esta en /ruta-de-nuestro-proyecto/igep/custom/) y lo pegamos en la misma ruta (cambiándole de nombre al que queremos que sea el nombre de nuestro custom) .

Una vez hecho eso modificamos las rutas dentro de los siguientes archivos para que

- /ruta-de-nuestro-proyecto/igep/custom/nuestroCustom/**include.php**
- /ruta-de-nuestro-proyecto/igep/custom/nuestroCustom/**include_class.php**

y donde haya una referencia a "*default*" la remplazamos por el nombre que le hemos dado al directorio de nuestro custom .

Por ultimo modificamos la directiva de configuración **<customDirName>** en el archivo /ruta-de-nuestro-proyecto/igep/**gvHidraConfig.inc.xml** indicando el nombre de nuestro directorio de custom (solamente el nombre, no la ruta completa):

```
<customDirName>nombre-dir-custom</customDirName>
```

y además especificamos el nombre de nuestra aplicación en el fichero /ruta-de-nuestro-proyecto/**gvHidraConfig.inc.xml**:

```
<applicationName>nombre_app</applicationName>
```

7.5.2. Correspondencias entre ventanas y código en el archivo aplicacion.css

Vamos a describir la correspondencia entre las partes de la ventana con cada una de los

selectores que se definen en el archivo /ruta de nuestro custom/css/**aplicacion.css**

- **.formularios:**

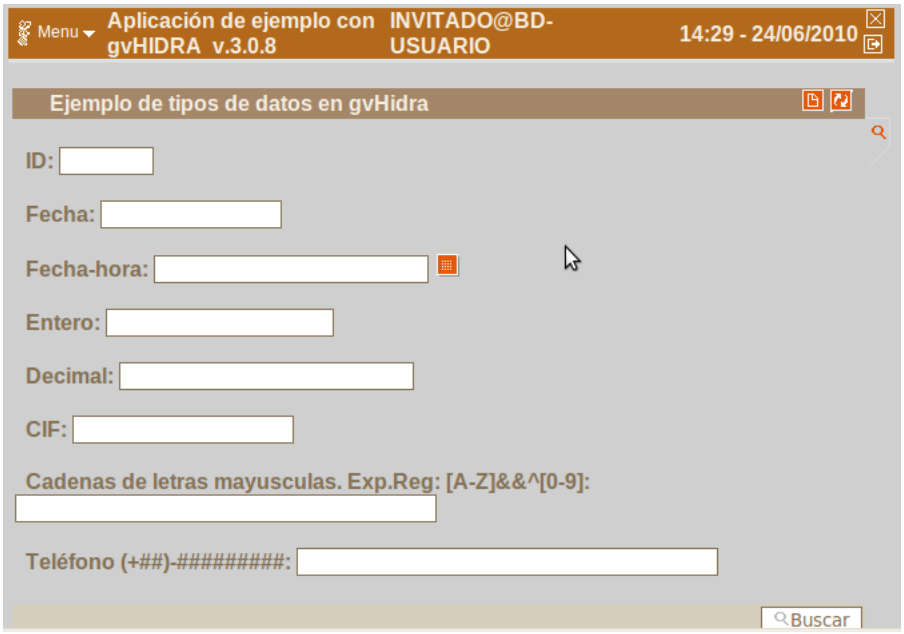
  Controla las características de los diferentes formularios de la aplicación (tanto en modo tabular como registro), en él, podemos cambiar el tamaño de las fuentes, el borde, el color, el background, ...

Aquí vemos el aspecto en una ventana con la definición por defecto:

Si cambiamos el font-size (tamaño), el font-weight(bold = negrita), y el color de fondo (background)

```
.formularios {
   font-family: Verdana, Arial, Helvetica, sans-serif;
   font-size: 16px;
   font-weight: bold;
   border: 0px;
   color: #857256;
   background-color: #cfcfcf;
}
```

Veríamos los siguientes cambios en el aspecto :



- **.text:**

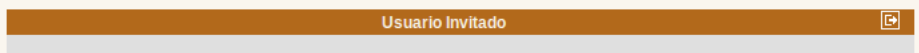
  Estilo que se aplica a la etiqueta que acompaña a un elemento de tipo radiobutton.

- **.menu y .submenu:**

  Estilo de los menús y submenús de la pantalla inicial.

- **.cabecerainicial:**

  Se corresponde con la barra superior de la página, en la que figura el nombre del usuario validado en el centro de la barra, y alineado a la derecha veremos el botón de salida de la aplicación si nos encontramos en la pantalla inicial, y en caso de que nos encontremos en una de las ventanas de la aplicación también veremos el botón de regreso a la pantalla principal.



  Probamos cambiar el color al negro, y la alineación de texto a izquierda:

```
.cabecerainicial {
   font-size: 12px;
   font-weight: bold;
   color: #000000;
   text-align: left;
   background-color: #B2691B;
}
```

- **.fondo_titaplic:**

  Corresponde al espacio que está debajo de la barra superior (*.cabecerainicial*) , y que está reservado para incluir el título y la descripción de la aplicación. Controlando el tamaño, alineación y el background del espacio.

```
.fondo_titaplic {
   width: 100%;
   height: 200px;
   vertical-align: bottom;
   background-attachment: fixed;
   background-repeat: no-repeat;
   background-position: center;
   background-color: #dfdfdf;
}
```

- **.titulo_aplicacion:**

  Controla las propiedades de la fuente del título que aparecerá en la pantalla principal, normalmente será el título de la aplicación.

```
.titulo_aplicacion {
  font-size: 80px;
  font-weight: bold;
  color: #2C658F;
}
```

- **.descrip_titaplic:**

  Corresponde al estilo que se le da a una pequeña descripción que podemos tener encima del titulo de la aplicación.

```
.descrip_titaplic {
  font-size: 20px;
  font-weight: bold;
  color: #000000;
}
```

  Imagenes de ejemplo para las tres clases anteriores :

  Imagen 1:

Imagen 2:

- **.fondo_inicio:**

  Básicamente se especifica el color del fondo de la aplicación, solo lo podremos ver reflejado en el espacio restante de la pagina inicial , que no esta ocupado por ventanas o menús.

- **.cabecera_modulos:**

  Controla color, alineación de texto, tamaño... de la barra que esta debajo del titulo de la aplicación.

- **.datos_cabecera:**

  Controla la presentación del texto incluido en la cabecera anterior (módulos principales, herramientas auxiliares, administración del sistema).

- **.menu_modulos:**

  Color, fondo, fuente tamaño de las entradas del menú de la aplicación.

```
.menu_modulos {
   font-size: 14px;
   font-weight: bold; //cambiamos de normal a negrita
   color: #000000;    //el color lo ponemos a negro
   text-decoration: underline; //cambiamos de none a underline (subrayado)
}
```

Imágenes de ejemplo:

Imagen 1:

Imagen 2:

Una vez dentro de una de las opciones del menú podemos ver, dentro de la ventana, una barra superior que contiene algunos datos de la sesión actual, y debajo esta el formulario para trabajar con la aplicación:

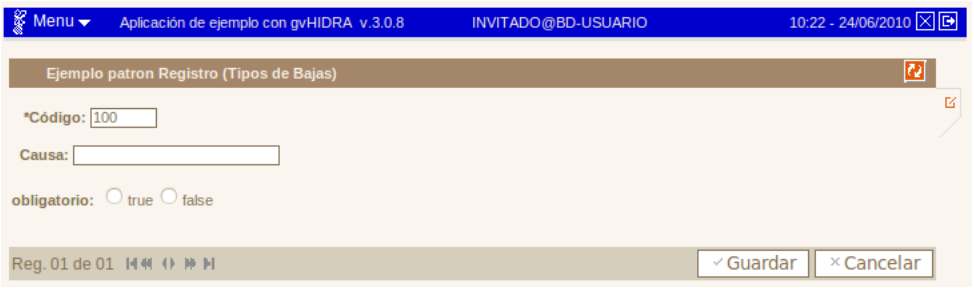
- **.barra_superior:**

  Definir el estilo de la barra superior, donde está el menú desplegable.

- **.txtcabecera:**

  Controla las características (tamaño, color, color-fondo, alineación de texto,...) de la barra superior de la ventana actual (no la ventana inicial).

```
.txtcabecera {
   color: #F5EEE1;
   background-color: #0000cc;  //cambiamos el fondo de la cabecera a azul.
}
```



- **.formulariosBarra:**

  Definir el estilo de la barra superior del panel, donde están los botones tooltip.

- **.barraSupPanel:**

  Controla las características de la barra superior de una ventana. Esta barra está situada directamente debajo de la barra superior de la ventana (*.txtcabecera*).

```
.barraSupPanel {
  font-weight: bold;
  color: #ffff33;  //cambiamos el color de texto a amarillo .
```

```
  background-color: #A48669;
}
```

- **.barraSupTitulo:**

  Estilo aplicable al título de la barra superior de un panel.

```
.barraSupTitulo {
  text-align: left;
}
```

- **.barraSupBotones:**

  Estilo aplicable a los botones tooltip de la barra superior de un panel.

```
.barraSupBotones {
    float:right;
    display:inline;
}
```

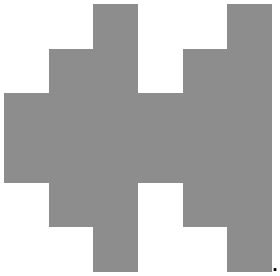
- **.barraInfPanel:**

  Controla las características de la barra inferior del panel. Es la barra donde se incluirán los botones (buscar, guardar, cancelar).

- **.paginador:**

  Controla la apariencia del paginador, cuando la consulta nos devuelve más datos de los que se pueden visualizar en una sola página, tanto cuando nos encontremos en modo registro como modo tabular.

- **.buttonPag:**

  Dentro de la apariencia del paginador, con este controlamos el aspecto de las imágenes que aparecen en el paginador, las imágenes que se utilizan para desplazarse entre páginas, por ejemplo  ,  .

- **.solapas:**

  Gestiona la presentación de las solapas en general, (color del fondo, tamaño, distancia entre la solapa y el borde del panel, ...).


```
.solapas {
  clear:both;
  border-left:2px solid #8F8F8F;
  border-bottom:2px solid #8F8F8F;
  border-top:1px none #D2D2D2;
}
```

- **.solapa:**

  Apariencia de las solapas inactivas, no seleccionadas.

- **.esqSolapa:**

  Apariencia de la esquina de la solapa (que tiene forma de triángulo).

- **.opcion:**

  Controlar la apariencia del titulo de una solapa inactiva (color, fuente, distancia a bordes, ...), no se refleja en la apariencia de la solapa activa.

- **.solapaActiva, .esqSolapaActiva y .infoFecha:**

  En este caso se controla la gestión de la solapa activa.

- **.boton:**

  Controla la apariencia de los botones (color, tamaño, fondo, ...), botones que aparecen en la barra inferior de la pantalla.

```
.boton {
   color: #ffffff;
   border: thin #857256 solid;
   background-color: #000000;
}
```

- **.boton_on:**

  Lo mismo que *.boton* pero solo cuando el ratón está sobre el botón.

```
.boton_on {
```

```
  color: #F5EEE1;
  border: thin #F5EEE1 solid;
  background-color: #0000cc;//fondo azul cuando pasamos el ratón por encima
}
```

- **.enlace:**

  Apariencia para los enlaces dentro de un campo de texto.

- **.editable y .noEditable:**

  Para controlar la apariencia de los campos de texto editables y no editables respectivamente.

```
.editable {
  color: #857256; //color de texto de campos de texto editables amarillo
  border: 1px solid #857256;
  background-color: #0000cc; // fondo azul
}
```

Imágenes de ejemplo:

Imagen 1:

Imagen 2:

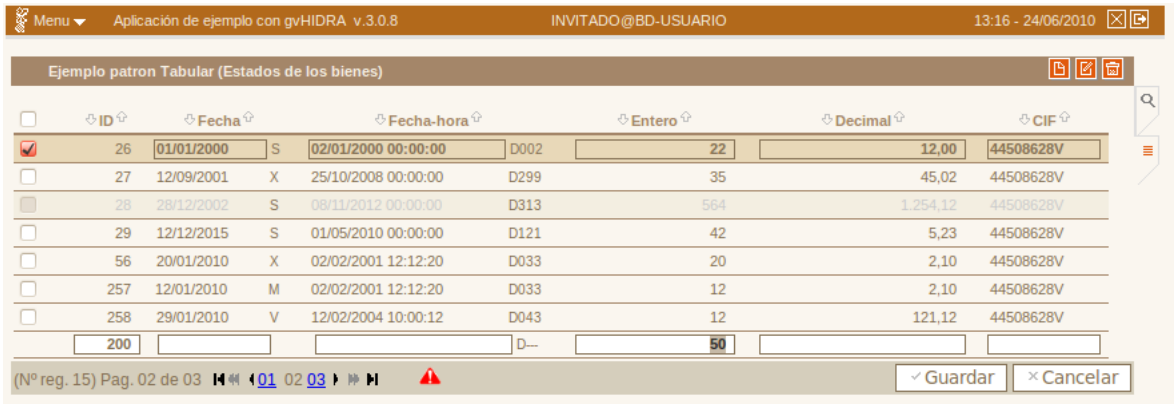
- **.nuevo, .modificable, .borrar:**

  Apariencia de un elemento del formulario (campo de texto, lista...), cuando nos encontramos en un panel tipo registro. Estilo .nuevo será el que corresponderá cuando el registro vaya a ser insertado, el .modificable, cuando el registro se vaya a modificar, y .borrar, por deducción, el registro estará marcado para ser borrado.

- **.tablaNuevo, .tablaModificar, .tablaBorrar:**

  Lo mismo que hemos indicado antes para un panel registro pero si nos encontramos en un panel tabular.

  Imagen inicial, y código de los cambios a aplicar:




```
.tablaModificar {
  font-weight: bold;
  color: #ffff00; //color del texto para una fila que se este modificando
  border: 3px solid #0000cc; //incrementamos el grosor del borde y el color de fondo.
}


.tablaBorrar {
  color: #ff0000; //rojo
  background-color: #ffff00;  //amarillo
}


.tablaInsertar {
  font-weight: bold;
  color: #ffffff; //cambiamos el color de texto para una fila nueva
  border: 1px solid #857256;
  background-color: #cc0033; //y también el color de fondo .
}
```

el resultado será:

- **.tablaEdi y .tablaNoEdi:**

  Los campos de una tabla que son editables / noEditables (background, color texto, ...).

- **.cargando:**

  Controla la apariencia del mensaje que aparece cuando una búsqueda o alguna operación tarda más en ejecutarse.

- **.registro_par, .registro_par input, .registro_par select, .registro_impar, .registro_impar input, .registro_impar select:**

  Estilos para definir filas pares e impares en un panel tabular.

- **.alerta, .aviso, .error, .sugerencia:**

Como se ha explicado en el capítulo 3 punto Mensajes y Errores [87], se clasifican los mensajes en cuatro tipos, estos estilos corresponden con cada uno de esos tipos.

```
.alerta {
  background-color: #FFE5C4;
}
```

- **.alerta input, .alerta select**

  .aviso input, .aviso select

  .error input, .error select

  .sugerencia input, .sugerencia select:

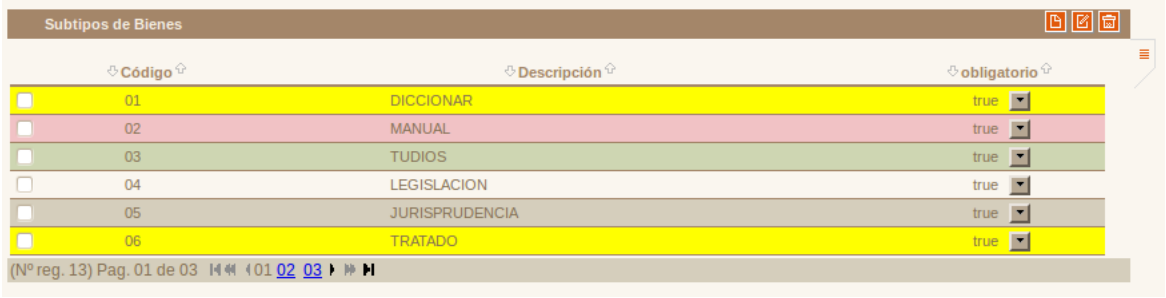
  Este aspecto se puede utilizar también para un marcado de aspecto de filas en un panel tabular, para ayudar a este marcado también se ha añadido para elementos de formulario que pueden aparecer en un tabular.

```
.alerta input {
  background-color: #FFE5C4;
}
.alerta select {
  background-color: #FFE5C4;
}
```

Imagen inicial y código de los cambios que se aplicarán:

Al cambiar el fondo de una fila, por ejemplo en modo alerta, vemos la diferencia en la primera y última fila de la tabla.

```
.alerta {
  background-color: #FFFF00;  //cambiamos a color amarillo .
}
.alerta input {
  background-color: #FFFF00;
}
.alerta select {
  background-color: #FFFF00;
}
```



- **.fila_on:**

  Controla la apariencia (p.ej su color de fondo) de una fila de una tabla cuando esta se seleccione.

- **.fila_on input, .fila_on select:**

  Estilo de una fila_on que se aplica a los elementos html .

- **.fila_over:**

  Apariencia de la fila cuando se pasa por el ratón encima.

- **.fila_over input, .fila_over select:**

  Estilo de una fila_over que se aplica a los elementos html .

- **.fila_borrada:**

  Apariencia (básicamente el color) de una fila marcada como borrada de una tabla.

- **.fila_ borrada input, .fila_ borrada select:**

  Estilo de fila borrada que se aplica a los elementos html .

- **.tabla_registros, .tabla_cabecera, .tabla_titulo, .columna_cabecera:**

  Definir el estilo de la tabla que engloba los registros.

- **.linea_cabezera_tabla y .linea_separa_filas:**

  Apariencia (grosor, color) de la línea que está debajo de los nombres de las columnas de una tabla y apariencia de las líneas entre filas de la tabla, respectivamente.

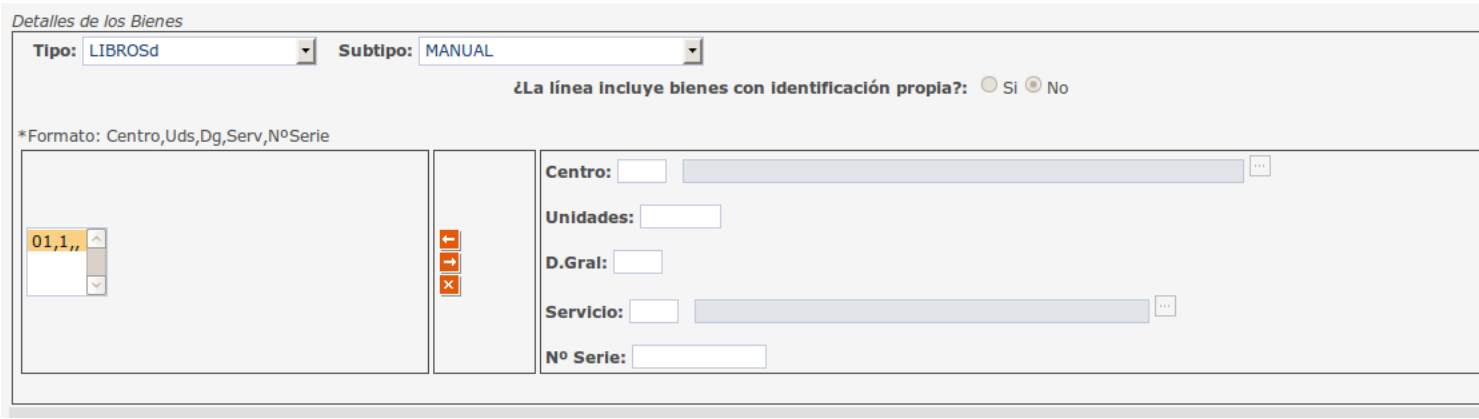
- **.cabecera_tabla, .tabla_titulo:**

  Apariencia de la cabecera y el título de una tabla dentro de un panel.

- **.grupoCampos:**

  Si necesitamos marcar un grupo de campos podemos utilizar este estilo.

```
.grupoCampos {
   color: #4E4E4E;
   border: 1px solid #4E4E4E;
   height: 18px;
}
```



- **.fondo_ (gris | azul | rojo ...) .avisocorto , .avisolargo y codigoerror:**

  Controlan la apariencia de los mensajes de alerta, aviso, sugerencia y error (colores de fondo básicamente), y tamaño de texto y color, y alineación de los diferentes tipos de aviso que se puedan lanzar.

- **.arbol:**

  Apariencia del panel inicial de un árbol (que está en la parte izquierda de la ventana).

- **.arbolIgep :**

  Controlar color, tamaño, background del nodo raíz del árbol.

- **.arbolIgep a , .arbolIgep a:link , .arbolIgep a:visited , .arbolIgep a:hover :**

  Controlar apariencia del nodo raíz dependiendo del su estado (visitado: visited , no visitado: link , y cuando se pasa el ratón por encima sin pinchar : hover).

- **.arbolIgepSeleccionado a , .arbolIgepSeleccionado a:link , .arbolIgepSeleccionado a:visited , .arbolIgepSeleccionado a:hover :**

  Lo mismo que en el caso anterior, pero solo se aplica cuando el nodo raíz ha sido ya seleccionado y expandido. Por ejemplo, cuando se pasa el ratón por encima del nodo raíz, una vez este ha sido seleccionado y expandido, se aplica entonces el código que está en la sección .arbolIgepSeleccionado a:hover , y no el de arbolIgep a:hover .

- **.nodoIgep1 y nodoIgep1Seleccionado (con sus posibles opciones a:link a:hover ...etc) :**

  Igual para el caso anterior (que se aplicaba solo al nodo raíz), pero se aplica en este caso solo a los nodo del árbol de primer nivel (solo los que aparecen al expandir el nodo raíz) .

  Imagen inicial y código de los cambios a aplicar:

Cambiamos el estilo de los nodos del nivel 1:

```
.nodoIgep1 {
  font-size: small;
  font-style: italic;
  color: #857256; //cambiamos el color a negro
  text-decoration: none;
}

.nodoIgep1 a:link {
  text-decoration: none;
  color: #000000; //color nodos del primer nivel no visitados a negro
}
```

y cambiamos el estilo cuando se pasa por el ratón por encima de nodos de primer nivel del árbol:

```
.nodoIgep1 a:hover {
  font-size: 20px ; //tamaño mas grande
  color: #0000cc; //se cambia de color
  font-weight: bold;
  text-decoration: underline; //y el nodo se subraya
}
```

- **.nodoIgep2 y nodoIgep1Seleccionado (con sus posibles opciones a:link a:hover ...etc) :**

  Igual que nodoIgep1, pero aplicado en este caso solo a los nodos del segundo nivel y posteriores del árbol.

- **.enlacebotonActivo :**

  Controlar la apariencia del botón activo (pulsado actualmente) de un panel maestro-nDetalles.

- **.enlaceboton , .enlaceboton:link , .enlaceboton:visit , .enlaceboton:hover :**

  Controlan la apariencia para el resto de botones no activos del panel maestro-nDetalles (*no visitados*: aplica la selección de link, *visitados*: aplica la de visited, *cuando pasamos el ratón por encima*: se aplica el código en la sección hover).

Capítulo 8. Bitacora de cambios aplicados a gvHidra

8.1. Historico de actualizaciones

En este documento se detallan los cambios realizados en cada versión.

8.1.1. Versión 3.1.1

(31-01-2011)

Errores solucionados:

- No aparece la imagen al desactivar los tooltip del detalle.
- Warnings en acceso al getAllTuplas.
- Error en documentación: imagen de mensaje confirmación.
- Notice por constantes no defenididas en gvHidraErrorHandlers.
- Las ventanas de confirmación no funcionan.
- Campo fecha con editable="nuevo" tiene el calendario activo siempre.
- Error al ordenar por fechas en un tabular.
- Ubicación correcta del templates en preproducción.

Mejoras aplicadas:

- Renombrar funciones en inglés (showMensaje y setAllTuplasAntiguas).
- Renombrar funciones en inglés (setParametrosBusqueda).
- Actualización proyecto jasper 4.0.0.2

8.1.2. Versión 3.1.0

(18-11-2010)

Errores solucionados:

- Problema con maestro-detalle cuando campos clave son listas.
- Error de comprobación de obligatorios en inserción.
- Error en plantillasTipo P1M1(FIL).
- accion de interfaz desde un radiobuton a otro radiobuton.
- Controlar longitud de la versión en el log de aplicaciones.

- Ajuste del debug para que funcione tambien en mysql y oracle.
- Problema setVisible con listas.
- La serialización de objetos gvHidraTimestamp no se hace correctamente.
- Problema con los mappings de accion particular guardar.
- Compatibilidad con PHP 5.3.
- Revisión del typeNIF para compatibilidad con PHP 5.3.
- Problema al modificar una lista, no guarda el valor elegido.
- En array de módulos dinámicos el índice para la descripción no sigue la estructura definida para los módulos.
- Error en tabulares con las listas/radios.
- Error que muestra en el oculto si la select de búsqueda falla.
- Error en las listas en un tabular cuando la consulta no devuelve datos.
- Problema de velocidad al seleccionar un registro en un tabular con selección única.
- Las listas no editables no envían el valor.
- Fallo en la dependencia de las VS al insertar en los patrones registro.
- Panel lis, con tipoListado a true muestra los campos ocultos.
- Problema después de insertar en tabla varios registros.
- Revisión de plantillas Maestro Detalle.
- Un campo tipo gvHidraString cuando tiene expresión regular no puede ser vacio.
- Validacion de tipos de datos antes de: buscar, saltar, volver y acciones particulares.
- La búsqueda en las ventanas de seleccion falla a partir de postgresql 8.3.
- No aparecen datos en la ventana de selección.
- No se comprueba bien en las fechas el último dia del mes.
- Error de generación de cabeceras de javascript.
- Warnings en IgepComunicacion::array_values_with_clone.
- No funcionan las rutas relativas en envio de correos en bloque.
- Error control de acceso módulos con valor.
- Error documentación acciones particulares.
- El CWUpLoad no marca que hay cambios cuando se actualiza.
- Error maestro detalle con tipos de datos.

- Error en el manejo de los datos por defecto en las listas (método addDefaultData).
- Tratamiento de excepciones en servidores de web services.
- Error plantilla mappings: referencia al panel de salida en iniciarVentana.
- No funciona la limpieza de campos en el filtro de las Busquedas (FIL).
- Maestro(LIS)-Detalle. Pérdida de la referencia del maestro si deseleccionamos el registro activo.
- En postgresql no va el like con campos numéricos (desde version 8.3).
- CWLista por defecto queda no editable.
- Error en la ordenación de las fechas en las tablas.
- El cero como tipo numerico no se transforma correctamente con IgepComunicaUsuario::prepararPresentacion.
- Inconsistencia en el rango de valores del queryMode.
- El tipo de dsn incorrecto no es detectado en la carga del xml de configuración.
- Limpiar la sesión al entrar a la aplicación.
- No tiene efecto el atributo dnsRef del elemento logSettings de la configuración.
- Problema con los nombres de las claves primarias en un tabular-registro.
- En _debugger.php habia una referencia absoluta a custom cit.gva.es.
- Si no existe el custom cit.gva.es se produce un error.
- Advertencia en IgepSession::hayModulo cuando no existen modulos dinámicos.
- Controlar que el tipo de mensaje en el debug sea numérico.
- Actualizar documentación del arbol.
- Actualizar documentación de los plugins.
- Actualizar documentación de patrón tabular-registro.
- Eliminar informacion innecesaria del REQUEST.
- Funcionamiento erroneo en la transformación de cadenas cuando está habilitado el magic_quotes_gpc.
- Error en el constructor ventanas seleccion. No admitia conexion alternativa.
- Error en la consulta tras insertar si tenemos campos sin matching.
- Cambio de servidores de produccion de postgres.
- Numero total de registros en un Tabular.
- Error en IgepDebug al pasarle objetos con print_r (con var_export va bien).
- Permitir cualquier caracter como alias de campo en ventanas de selección.

- Añadir rollback al finalizar las conexiones a BBDD.
- Fallo en el seleccionarTodo de las tablas.
- En oracle no funciona el empezarTransaccion, luego siempre está funcionando con autocommit.
- Error en el setVisible de los radios.
- Actualización a jasper 3.0.0.3.
- En la css del calendario hay referencia a imagen que no existe.
- Problema con las constantes MDB2 en la conexion.
- En ventanas de selección no se puede buscar usando comilla simple.
- El tipo de la columna tipo de la tabla tcmn_errlog cambia de varchar a numeric de 2.
- Revisar documentación de IgepConexion->prepararOperacion.
- Plugin Radio: no dispara modificación ni funciona obligatorio.
- El boton de limpiar campos no actua sobre los Radios.
- Desactivar listas.
- Los radiobutons no funcionan bien con el tabindex.
- Crear métodos para el acceso al filtro busqueda y edición.
- Inserción en tabla desde búsqueda.
- IgepSession los métodos de acceso a datos no devuelven formato PHP.
- Ejecución innecesaria del calculo de detalles erroneo cuando el maestro es vacío.
- Botón calendario se activa cuando es editable=false.
- Error al desconectar una conexion.
- Parametro openWindow de CWBoton: busca blanco en url incorrecta.
- Ventanas de seleccion con dependencia debil: error al introducir parámetro búsqueda.
- Error en plantilla de plantilla-P1M1(EDI).tpl, plantilla-P1M1(LIS).tpl.
- CWSelector no funciona en el panel de búsqueda.
- Error documentacion listas.
- Problema con el parámetro "numCaracteres" del plugin CWLista.
- Error escapado en las acciones de interfaz.
- Error en prepararOperacion, cuando no tiene tipo no escapa como TIPO_CARACTER.
- Acciones de interfaz en clase gvHidraForm.

- Eliminar warning en linea 281 de gvHidraForm_DB.
- Error al fallar la validacion de expresión regular.
- No funciona la tabulacion en campos editable =nuevo.
- Mal funcionamiento de la propiedad tabindex en los tabulares.
- Error conexiones persistentes en PostgreSQL.
- Error botontooltip lanzando acciones de interfaz.
- Actualizar clase IgepPeticiones del custom.
- Error con la contrabarra \ cuando utilizamos Oracle.
- Error documentacion setQueryMode en PHPDoc.

Mejoras aplicadas:

- IgepComunicacion::setAllTuplas acaba la ejecución si no recibe un array.
- Cambiar atributo dnsRef en ficheros gvHidraConfig.inc.xml por dsnRef.
- Configurar parámetros de la sesion para mejorar la seguridad.
- Fijar el encoding a latin1 en la clase cliente de web services.
- Nuevos métodos en el servidor de web services para tratar la codificación y los soap_fault.
- Acceso desde la css al estilo de la pantalla de entrada.
- Uso de inserciones preparadas en el debugger.
- En postgresql ya no es necesaria la funcion concat para la búsqueda en las ventanas de selección.
- Compatibilidad de tests unitarios con PHPUnit 3.3.
- Creación de plantillas base para maestro-ndetalles.
- Poner el enableServerValidation como deprecated.
- Restringir métodos de web services a un conjunto de credenciales.
- Se ha reemplazado el uso de ConfIgep::es_desarrollo por nuevas propiedades en gvHidraConfig.inc.xml.
- Se incluye el ignore para subversión en la plantilla de proyecto.
- Inicializar estado del framework en la ejecución de los tests.
- Quitar los dsn de la pantalla del debugger.
- En servidores de web services usar el login de la credencial como usuario en el debug.
- Nuevo método formatSOAP en gvHidraTimestamp para formatear fechas en web services.
- El atributo customDirName sólo se permite cambiar en xml de gvHidra y de la aplicación. No en la carga dinámica ni el el xml del custom.

- Cambio del comportamiento de la búsqueda en inserción de maestro.
- Creación de plantilla para el manual de usuario de la aplicación.
- Añadir en la web las versiones de jasper separadas de la plantilla de aplicación.
- Se añade a la documentación una relación de errores conocidos.
- Permitir parametrizar el comportamiento tras realizar una inserción.
- Mantener el valor de los campos del filtro, despues de buscar.
- Unificación parámetros plugins.
- Soporte a sentencias SQL preparadas.
- Uso interno de metodos MDB2 para empezar y acabar transacciones.
- Mostrar los mensajes del log de apache en el debug.
- Parametrizar el tamaño de ventana de selección.
- Revisión de los métodos para obtener el dsn y la conexión en una clase manejadora y en IgepConexion.
- Seleccionar todo el contenido al entrar en un campo de texto.
- Eliminar clase ConfIgep.
- Funcionamiento dinámico del tabIndex.
- Revision de clases de gvHIDRA en la carga dinámica.
- Nuevo método para poder formar condiciones siguiendo el mismo queryMode definido en el formulario.
- En las búsquedas descartar siempre carácteres especiales y no distinguir por mayúsculas.
- Posibilidad de definir una carpeta temporal para almacenar las sesiones.
- Mensajes de confirmación.
- Crear conexion con el registro de Salida.
- Poder cambiar el texto del mostrarEspera (actualmente Cargando...).
- Actualizar una imagen desde otro campo.
- Documentar el uso de los external.
- Actualizar a versión jasper 3.0.0.4.
- Vincular las imagenes al custom.
- Adaptar plugins para facilitar la visualización del custom de Sanidad.
- Css en menús de pantalla de entrada.
- Crear dependencia débil en las listas .

8.1.3. Versión 3.0.11

(31-01-2011)

Errores solucionados:

- Error al fallar la validacion de expresión regular.
- No funciona la tabulacion en campos editable ="nuevo".
- Mal funcionamiento de la propiedad "tabindex" en los tabulares.
- Error conexiones persistentes en PostgreSQL.
- Error botontooltip lanzando acciones de interfaz.
- Actualizar clase IgepPeticiones del custom cit.gva.es.

Mejoras aplicadas:

- Css en menús de pantalla de entrada.

8.1.4. Versión 3.0.10

(06-08-2010)

Errores solucionados:

- Maestro(LIS)-Detalle. Pérdida de la referencia del maestro si deseleccionamos el registro activo
- Actualizar documentación de los plugins
- Error documentacion listas
- Problema con el parámetro "numCaracteres" del plugin CWLista
- Error escapado en las acciones de interfaz
- Error en prepararOperacion, cuando no tiene tipo no escapa como TIPO_CARACTER
- Acciones de interfaz en clase gvHidraForm
- Eliminar warning en linea 281 de gvHidraForm_DB

Mejoras aplicadas:

- Documentar el uso de los external
- Actualizar a versión jasper 3.0.0.4
- Vincular las imagenes al custom
- Adaptar plugins para facilitar la visualización del custom de Sanidad

8.1.5. Versión 3.0.9

(03-06-2010)

Errores solucionados:

- Actualizar documentación de patrón tabular-registro.
- El boton de limpiar campos no actua sobre los Radios.
- Inserción en tabla desde búsqueda (atención: siempre con accion="nuevo" en el CWBotonToolTip).
- IgepSession los métodos de acceso a datos no devuelven formato PHP.
- Ejecución innecesaria del cálculo de detalles erroneo cuando el maestro es vacío.
- Error al desconectar una conexión de PostgreSQL.
- Parametro openWindow de CWBoton: busca blanco en url incorrecta.
- Error de validacion en la demo.
- No aparecen los mensajes del postBuscar del Maestro.
- Warning en el método checkData cuando la matriz de datos es vacia.
- Ventanas de seleccion con dependencia debil: error al introducir parámetro búsqueda.
- No se actualiza la tupla actual tras una busqueda en maestro detalle.
- En maestro - detalle. En una accion particular del maestro que ejecuta un refreshSearch (deep) hace que el preRecargar y postRecargar del detalle pierdan la fila actual.
- Error en plantilla de plantilla-P1M1(EDI).tpl, plantilla-P1M1(LIS).tpl.
- Ordenacion del maestro tabular en patrones maestro detalle.
- CWSelector no funciona en el panel de búsqueda.

Mejoras aplicadas:

- Mensajes de confirmación.
- Poder cambiar el texto del mostrarEspera (actualmente 'cargando...')
- Acciones de interfaz: actualizar una imagen desde otro campo.
- Mejora del dummy: dos fuentes de datos para el tabular-registro.
- Reutilizacion de conexiones a BBDD (sólo conexiones PostgreSQL).
- Documentación sobre conexiones y llamadas a procedimientos almacenados.

8.1.6. Versión 2.2.13

(11-06-2008)

Errores solucionados:

Mejoras aplicadas:

8.2. Como migrar mis aplicaciones a otra versión de gvHidra

En este documento apuntaremos las acciones a realizar en una aplicación que utilice IGEP, para pasar de una versión a otra. Cada vez que hagamos alguna modificación en igep que vaya a afectar a las aplicaciones que lo utilicen, lo apuntaremos aqui. Este documento está orientado a Informáticos.

8.2.1. Versión 3.1.0

23-11-2010

- Si se está utilizando de forma complementaria al framework el proyecto jasper para creación de listados, en esta versión ya se puede utilizar la versión 'jasper-3_0_0_4'.
- Si el método **IgepComunicacion::setAllTuplas** no recibe un array corta la ejecución. Asegurarse que si se le pasa la salida del método consultar, hayan registros.
- Si se ha definido algún método nuevo de autenticación, en método autenticate añadir **IgepSession::session_start($p_apli, false);** antes de crear instancia de Auth.
- Si se usan clientes de web services con la clase **IgepWS_Client** y no se ha fijado la opción de codificación, ahora se fija a **latin1**. Con esto ya no es necesario usar los métodos uff8_encode/utf8_decode para convertir los datos.
- **(RECOMENDADO)** El atributo **logSettings** ya no se fija en la carga dinámica; añadirlo al **gvHidraConfig.inc.xml** según el nivel de log deseado en cada servidor donde se despliegue la aplicación.
- **(RECOMENDADO)** Si se usan fechas en servidores de web services se recomienda generarlas usando el método formatSOAP de **gvHidraTimestamp**.
- Cambiamos el nombre del método addCamposClave por **setPKForQueries**.
- El método ConfigFramework::setCustomDirName ya no existe. Si se ha definido algún custom nuevo, hay que actualizar la DTD porque ya no se puede fijar el atributo **customDirName** (en los xml de la aplicación y del framework si se puede). Podéis copiar la DTD correcta del xml de cualquier custom.
- Los métodos de **ConfIgep** *formatoFecha, formatoNumero y formatoFechaNegocio* ya no existen.
- Los métodos de **IgepComunicaUsuario** *strtotime_es y timetostr_es* ya no existen.
- El método *ConfIgep::es_desarrollo* ya no existe. Si hay que hacer alguna acción dependiente del servidor, intentar configurar el xml para cada servidor, o si no hay más remedio usar el nombre del host para decidir. Cuestiones relacionadas:
  - En el servidor de producción ya no se fija el error_reporting (E_USER_ERROR | E_USER_WARNING | E_USER_NOTICE). Se hará como esté definido en el servidor PHP.
  - Fijar el parámetro **<reloadMappings>false</reloadMappings>** en **producción**, ya que por defecto está habilitado.
  - Fijar el parámetro **<smartyCompileCheck>false</smartyCompileCheck>** en **producción**, ya que por defecto está habilitado.
- Actualizar la **DTD** del fichero **gvHidraConfig.inc.xml**. Podéis copiar la DTD correcta del xml de la plantilla de aplicación. Cambios realizados:

  - Se fija el rango del atributo status de queryMode de 0 a 2. Si se ha fijado este elemento comprobar que es el deseado. El valor 3 ya no existe por lo que si se estaba usando (en ConfigFramework::setQueryMode, gvHidraForm_DB::setTipoConsulta o gvHidraSelectionWindow::setQueryMode) cambiar por valor 1.
  - Cambia el atributo dnsRef de logSettings por el atributo dsnRef.
  - Se añade el elemento opcional temporalDir, usado para fijar la ubicación del fichero de sesión. (CIT: Es imprescindible para uso en producción)
- Cambios en las listas. La clase IgepLista pasa a llamarse **gvHidraList**, podemos definir una fuente de datos externa. Pasos de migración obligatorios:
  - Reemplazar en todo el código cualquier ocurrencia de *"IgepLista" -> "gvHidraList"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"$this->addLista" por "$this->addList"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->marcarSeleccionado" por "->setSelected"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->setSeleccionado" por "->setSelected"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->addOpcion" por "->addOption"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->deleteOpcion" por "->deleteOption"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->setDefList" por "->setList_DBSource"*.
- Cambios en las ventanas de selección. La clase IgepVentanaSeleccion pasa a llamarse **gvHidraSelectionWindow**, todas las ventanas de selección tienen matching, podemos definir fuentes de datos externas, podemos fijar una tpl a partir de la cual mostrar dicha ventana.
  - Reemplazar en todo el código cualquier ocurrencia de *"IgepVentanaSeleccion" por "gvHidraSelectionWindow"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"$this->addVentanaSeleccion" por "$this->addSelectionWindow".*
  - *Añadir addMatching*. Para poder reutilizar las ventanas de selección en diferentes paneles, ahora incorporan Matching. Esto quiere decir, que tendrán un nombre los campos resultado y nosotros podremos asociarlos a nuestros campos de la tpl. A efectos de migración, esto supone que, el tercer parámetro de las antiguas llamadas a IgepVentanaSeleccion se transforma en una o varias llamadas al método addMatching.

```
// VERSIÓN ANTERIOR
$codper = new IgepVentanaSeleccion("codper","PERSONAS",array("codper","nom"));
$this->addVentanaSeleccion($codper);

// VERSIÓN 3.1.1
$codper = new gvHidraSelectionWindow("codper","PERSONAS"); //Eliminamos el array.
$codper->addMatching('codper','codper'); //Creamos tantas llamadas al metodo addMatching como elementos del array
$codper->addMatching('nom','nom');  // Nota: los dos parametros serán el mismo para que funcione como en la version 3.0.X
$this->addSelectionWindow($codper);
```

  - Reemplazar en todo el código cualquier ocurrencia de *"->setDefVS" por "->setSelectionWindow_DBSource"*.
- Reemplazar en todo el código cualquier ocurrencia de **"->setDependencia" por "->setDependence"**.
- Cambio del nombre de la clase de cheks. Reemplazar en todo el código cualquier ocurrencia de "IgepCheckBox" por "**gvHidraCheckBox**"

8.2.2. Versión 3.0.0

(26-6-2009)

- Si se está utilizando de forma complementaria al framework el proyecto jasper para creación de listados, en esta versión ya se puede utilizar la versión 'jasper-3_0_0_1'.

- Repasar la DTD del fichero gvHidraConfig.inc.xml de la aplicación para incluir las modificaciones relativas a la conexión a BD. [Basta copiar el bloque de texto desde la linea 64 a la 85 aprox. del fichero gvHidraConfig.inc.xml, y pegarlo en el fichero gvHidraConfig.inc.xml de cada aplicación]. (Es **IMPRESCINDIBLE** si se quieren utilizar reports jasper contra BDs oracle, además de actualizar a la última versión del proyecto jasper). El cambio a una nueva versión del proyecto jasper, puede presentar problemas de incompatibilidad hacia atrás, por lo que es recomendable recompilar TODOS los reports. Para más información, cunsultar la documentación del proyecto jasper [http://zope.coput.gva.es/proyectos/jasper].

- Cambio de las clases de herencia de Negocio. Se han creado varias clases de las cuales el programador podrá heredar dependiendo del tipo de form que esté implementado. Esto implica cambiar la herencia de todas las clases manejadoras de IgepNegocio a una de las siguientes clases:

  - gvHidraForm: Clase que se utiliza para el manejo de paneles sin conexión a BD.

  - gvHidraForm_DB: Clase que se utiliza para el manejo de paneles con conexión a BD.

  - gvHidraForm_dummy: Clase que se utiliza para realizar prototipos de pantalla.

  La recomendación en este apartado es cambiar la herencia de IgepNegocio a gvHidraForm_DB excepto cuando el panel claramente, no tiene acceso a BD.

  Al cambiar la herencia, la llamada al padre debe ser parent::__construct en lugar de parent::IgepNegocio.

- Cambio en el formato de los datos: ahora siempre se pueden manejar los datos en formato negocio (el punto como separador decimal para números, y para fechas instancias de gvHidraTimestamp). Detalles:

  - Los métodos de ConfIgep formatoFecha y formatoNumero ya no son necesarios y se consideran obsoletos, puesto que el programador siempre trabaja en formato negocio. Si en algún momento se necesita usar los formatos de presentación, se puede usar IgepComunicaUsuario::prepararPresentacion.

  - los métodos de ConfIgep transformaNumero y formatoFechaNegocio ya no son necesarios y se consideran obsoletos, puesto que el programador no debe recibir en ningún caso formatos de presentación. Si en alguna situación son necesarios usar IgepComunicacion->transform_User2FW, y reportarlo al registro de soporte. El método formatoFechaNegocio puede ser útil todavía cuando se necesite validar las fechas de entrada.

  - las fechas vienen ahora como instancias de gvHidraTimestamp (basado en DateTime [http://es.php.net/manual/en/class.datetime.php]). Ver la ayuda para saber como operar con esos objetos. Hay que evitar usar fechas timestamp, especialmente si se tienen fechas anteriores a 1970. Habría que reemplazar las funciones date, time, mktime, strtotime, ... Hay métodos equivalentes para esos casos, sin la restricción del timestamp.

  - ConfIgep::comparaFechas() y IgepComunicaUsuario::comparaFechas() ya no existen; en su lugar usar gvHidraTimestamp::cmp que compara fechas en negocio. Si por algún motivo se tienen fechas en formato User, convertirlas previamente con IgepComunicacion->transform_User2FW.

  - IgepComunicaUsuario::timetostr_es es obsoleta. Si se necesita convertir una fecha del FW a formato User utilizar gvHidraTimestamp->formatUser().

  - IgepComunicaUsuario::strtotime_es es obsoleta. Si se necesita convertir una fecha del usuario a formato FW utilizar IgepComunicacion::transform_User2FW.

- En las credenciales de los servidores de web services, la contraseña hay que almacenarla con hash. Para ello usar el formulario en igep/include/igep_utils/protectdata.php para obtener los hash de las contraseñas, y guardar estas últimas en un lugar seguro, fuera de la aplicación. Esto NO AFECTA en ningún modo a los clientes de web services.

- Los radios cambian totalmente. El parámetro radio del plugin desaparece y ahora se configura a través del método setRadio de la clase gvHidraList. Se tiene que incluir dataType a la definición del plugin porque es por donde le llega la información.
- Listas múltiples. El parámetro múltiple del plugin desaparece y ahora se configura a través del método setMultiple de la clase gvHidraList. Se tiene que incluir dataType a la defición del plugin porque es por donde le llega la información.
- Los checkbox cambian radicalmente. Ahora se definen en negocio mediante la clase gvHidraCheckBox. Se deben incluir al panel. En el plugin se elimina el parámetros valorSi y valorNo que vendrán a partir del dataType.
- Remplazar las referencias a métodos obsoletos por los correctos:
  - setCampo -> setValue (sustituir setCampo por setValue).
  - getCampo -> getValue.
  - setSeleccionado -> setSelected sólo en las **acciones de interfaz y acciones genericas.**
  - getCampoDisparador -> getTriggerField
- (RECOMENDADO) Si teneis algún servidor de web services que tenga varias credenciales, ahora podéis tratarlos sin redefinir el método 'checkCredential', sino pasando un array de id's al constructor del servidor.
- (IMPORTANTE) A partir de la versión 3, se validan los campos antes de realizar ciertas acciones. Concretamente antes de buscar, saltar, regresar de salto, acciones particulares y acciones de actualización de BD (insertar, modificar borrar y operarBD). Hasta esta versión sólo se realizaba esta validación al realizar acciones de de actualización de BD. Esto significa que si tenemos el mismo campo en un panel fil y lis, si es obligatorio, lo será en los dos. Es importante tener esto en cuenta en todos los campos con definición de tipo de datos.
- Acceso a propiedades str_select, str_where, str_orderBy, str_selectEditar, str_whereEditar y str_orderByEditar. Se deben utilizar los nuevos métodos:
  - setSelectForSearchQuery: se debe usar este método para dar valor a la propiedad str_select.
  - setWhereForSearchQuery: se debe usar este método para dar valor a la propiedad str_where.
  - setOrderByForSearchQuery: se debe usar este método para dar valor a la propiedad str_orderBy.
  - setSelectForEditQuery: se debe usar este método para dar valor a la propiedad str_selectEditar.
  - setWhereForEditQuery: se debe usar este método para dar valor a la propiedad str_whereEditar.
  - setOrderByForEditQuery: se debe usar este método para dar valor a la propiedad str_orderByEditar.

Parte V. Apendices

Tabla de contenidos

Apéndice A. FAQs, resolución de problemas comunes

A.1. ¿?

A.2. ¿?

A.3. ¿?

Apéndice B. Pluggins, que son y como usarlos en mi aplicación

B.1. Documentación Plugins gvHidra

Lista de Pluggins

24.CWSolapa [196]

25.CWTabla [197]

26.CWUpLoad [198]

27.CWVentana [199]

B.1.1. CWArbol

Crea paneles vinculados a una estructura jerárquica.

En la parte izquierda dibuja el árbol, y el panel de la derecha corresponde al nodo seleccionado.

- **Plugins que pueden contener a CWArbol**
  - Padres
    - CWMarcoPanel
- **Plugins que puede contener CWArbol**
  - Hijos
    - CWPanel

Tabla de argumentos de CWArbol

Nombre	Tipo	¿Opcional?	Descripción	Valores
estado	alfanumérico	false	Indica el estado del panel (activo o desactivado) cuando se carga la pantalla. Normalmente es una variable smarty ($estado_edi) que será sustituida por el php correspondiente del views.	Puede tomar los siguientes valores: • *on*: visible y activo • *off*: visible e inactivo • *inactivo*: no visible e inactivo
arbol	alfanumérico	false	Estructura definida para el árbol (fichero xml). Será una variable smarty ($smty_objArbol) que es sustituida por IgepPanel.	
titulo	alfanumérico	true	Dará el título al panel que contiene el árbol.	
ancho	numerico	true	Define el ancho del panel árbol en porcentaje, por defecto es un 25%.	

Ejemplos de uso del plugin CWArbol:

Árbol con dos paneles diferentes. La variable **$smty_panelVisible** es sustituida por **IgepPanelArbol**, que le asignará el valor del tipo de nodo que se ha seleccionado, coincidrá con el nombre que se le pase como primer parámetro a la función **setNodoPanel()**.

```
{CWArbol estado=$estado_edi arbol=$smty_objArbol}
```

```
 {if $smty_panelVisible == "ANYO" || $smty_panelVisible == "FACTURA"}
   {CWPanel
                    ..............
                {/CWPanel}
 {else}
                {CWPanel
                    ..............
                {/CWPanel}
 {/if}
{/CWArbol}
```

Lista de plugins [168]

B.1.2. CWAreaTexto

Equivalente al TEXTAREA de HTML.

- **Plugins que pueden contener a CWAreaTexto**
  - Padres
    - CWContenedor
    - CWFila
    - CWFicha
    - CWSelector
    - CWSolapa
- **Plugins que puede contener CWAreaTexto**
  - Hijos
    - El plugin CWAreaTexto es una hoja (no contiene otros plugins)

Tabla de argumentos de CWAreaTexto

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
longitudMinima	entero	true	Indica el número de carácteres mínimos que debe introducirse en el campo. Si no se alcanza el número mínimo de caracteres a introducir, se muestra un mensaje de ALERTA.
longitudMaxima	entero	true	Indica el número de carácteres máximos que pueden introducirse en el campo. Si se sobrepasa dicho valor, se muestra un mensaje de ALERTA.
value	alfanumérico	true	Valor por defecto que aparecera en el campo.
editable	alfanumérico	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el

Nombre	Tipo	¿Opcional?	Descripción
			valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWAreaTexto sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
cols	entero	true	Especifica el número de columnas que tendrá el textarea. En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.
rows	entero	true	Especifica el número de filas que tendrá el textarea.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Fija el texto descriptivo que acompaña a un campo (etiqueta). Si además aparece el argumento/parámetro "obligatorio" a true, se le añade un * que indicará que el campo es obligatorio de rellenar. El texto que se incorpora como etiqueta, será también el que se utilice en los mensajes de comprobación de cmapos obligatorios, etc...
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
actualizaA	alfanumérico	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
visible	booleano	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/ invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.

Ejemplos de uso del plugin CWAreaTexto:

Ejemplo: Declaración de CWAreaTexto.

```
{CWAreaTexto nombre="comentario" textoAsociado="Expedientes" cols="20" rows="10"}
```

Lista de plugins [168]

B.1.3. CWBarra

Dibuja la barra superior común a todas las ventanas, comprende el menú de la aplicación, una descripción de la pantalla o formulario donde nos encontramos (la entrada de menú correspondiente). El usuario que se ha validado en la aplicación, y por último la hora y fecha local del PC desde el que se ejecuta el navegador Web.

- **Plugins que pueden contener a CWBarra**

- Padres
  - CWVentana

- **Plugins que puede contener CWBarra**
  - Hijos
    - CWMenuLayer

Tabla de argumentos de CWBarra

Nombre	Tipo	¿Opcional?	Descripción
usuario	alfanumérico	true	Fija el nombre de usuario en la Barra. Variable smarty ($smty_usuario) asignada internamente.
codigo	alfanumérico	true	Fija el nombre de la aplicacion en la barra Variable smarty ($smty_codigo) asignada internamente.
customTitle	alfanumérico	true	En la barra superior de color azul, a la izquierda de la fecha y hora, se ha reservado un pequeño espacio para poder incluir un texto personalizado. La asiganción se realiza a través de la variable smarty ($smty_customTitle) asignada internamente por gvHidra en IgepPantalla

Ejemplos de uso del plugin CWBarra:

Ejemplo: Una barra con Menú (plugin **CWMenuLayer**).

```
{CWBarra usuario="$smty_usuario" codigo="$smty_codigo" customTitle=$smty_customTitle}
  {CWMenuLayer name="$smty_nombre" fichero="$smty_fichero"}
{/CWBarra}
```

Lista de plugins [168]

B.1.4. CWBarraInfPanel

Dibuja la barra inferior de un panel. Contiene los botones que realizan acciones sobre la capa de persistencia (botones inferiores según la guía de estilo).

- **Plugins que pueden contener a CWBarraInfPanel**
  - Padres
    - CWPanel

- **Plugins que puede contener CWBarraInfPanel**
  - Hijos
    - CWBoton

Tabla de argumentos de CWBarraInfPanel

El plugin CWBarraInfPanel no tiene argumentos

Ejemplos de uso del plugin CWBarraInfPanel:

Ejemplo: Declaración de CWBarraInfPanel.

```
{CWBarraInfPanel}
  {CWBoton imagen="41" texto="Guardar" class="boton" funcion="guardar"}
{/CWBarraInfPanel}
```

Lista de plugins [168]

B.1.5. CWBarraSupPanel

Dibuja lo que en un panel será la cabecera de la ventana donde irá el título de esa ventana.

- **Plugins que pueden contener a CWBarraSupPanel**
  - Padres
    - CWPanel
- **Plugins que puede contener CWBarraSupPanel**
  - Hijos
    - CWBotonTooltip

Tabla de argumentos de CWBarraSupPanel

Nombre	Tipo	¿Opcional?	Descripción
titulo	alfanumérico	true	Establece la descripción de la acción del panel.

Ejemplos de uso del plugin CWBarraSupPanel:

Ejemplo: Declaración de CWBarraSupPanel.

```
{CWBarraSupPanel titulo="BUSCAR ESTADOS"}
  {CWBotonTooltip imagen="04" titulo="Limpiar campos" funcion="limpiar"}
{/CWBarraSupPanel}
```

Lista de plugins [168]

B.1.6. CWBoton

Equivalente al BUTTON de HTML. Generalmente, el uso de estos botones será para acceso a la capa de persistencia.

- **Plugins que pueden contener a CWBoton**
  - Padres
    - CWBarraInfPanel
- **Plugins que puede contener CWBoton**
  - Hijos
    - El plugin CWBoton es una hoja (no contiene otros plugins)

Tabla de argumentos de CWBoton

Nombre	Tipo	¿Opcional?	Descripción	Valores
name	alfanumérico	true	Nombre para identificar la instancia del componente, y se utilizara como texto del botón.	
class	alfanumérico	true	Indica el estilo css del botón que se genera. Por defecto tiene el marcado por la guía de estilo ('boton'). En caso de querer cambiarlo se debería incluir en la css.	
imagen	alfanumérico	true	Nombre del fichero que contiene la imagen que queremos que aparezca en el botón, junto al texto, pero sin añadirle la extensión. Actualmente, estas imágenes deben estar siempre en formato "gif"	
mostrarEspera	boolean	true	Su valor por defecto es false. Si la accion que desencadena el boton requiere un tiempo de proceso considerable, es conveniente utilizar este parámetro fijándolo a 'true'. Con ello conseguimos que durante el procesado de la acción que desencadena el botón, la interfaz permanezca bloqueda y se muestre al usuario un mensaje que indica que espere.	
funcion	alfanumérico	true	Este parametro se utiliza por si queremos añadirle funcionalidad extra al botón. En caso de utilizarse no debe añadirse la palabra reservada "javascript:".	
accion	alfanumérico	true	Especifica la acción que realiza el botón	Puede tomar los siguientes valores: • *guardar*: Ejecuta la acción del panel. • *cancelar*:Cancela la acción del panel. • *saltar*: La acción es un salto. • *volver*: Se retorna de un salto. • *listar*: La accion invoca a un listado. • *cancelarvs*: Para un botón Cancelar de una ventana emergente.

Nombre	Tipo	¿Opcional?	Descripción	Valores
				• *aceptarvs*: Sólo para el botón Aceptar de la ventana de selección. • *particular*: El botón realiza una función que no es genérica. En este caso son obligatorios los parámetros "*id*" y "*action*". "a*ction*": acción particular que se ejecutará en la clase manejadora (método accionesParticulares). "id": identificador del botón. Si queremos que el botón funcione en cuanto a visibilidad como los guardar/ cancelar pondremos el parámetro visible = "false".
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio, y tratar su visibilidad posteriormente desde negocio.	
id	alfanumérico	true	Identificador para el botón, sólo en el caso de que la acción sea "particular".	
openWindow	boolean	true	Indica si queremos que la acción del botón la ejecute en una ventana diferente.	
action	alfanumérico	true	Acción que queremos que se ejecute al pulsar el botón y que sea diferente a la de por defecto del panel.	
texto	alfanumérico	true	Este parametro se utiliza para visualizar la ayuda en linea al situar el puntero del ratón sobre el componente.	
confirm	boolean	true	Con este parámetro aparecerá una ventana emergente de confirmación para ejecutar la acción del panel.	
filaActual	entero	true	Nos indica la fila seleccionada, **SÓLO** se utiliza en la **VENTANA DE SELECCIÓN** donde es OBLIGATORIO.	
formActua	alfanumérico	true	Parámetro OBLIGATORIO **SÓLO** cuando es un botón de la **VENTANA DE SELECCIÓN**. Indica el nombre del formulario en el que se devolverán	

Nombre	Tipo	¿Opcional?	Descripción	Valores
			los valores elegidos en la ventana de selección.	
panelActua	alfanumérico	true	Parámetro OBLIGATORIO **SÓLO** cuando es un botón de la **VENTANA DE SELECCIÓN**. Indica el nombre del panel en el que se devolverán los valores elegidos en la ventana de selección.	
actuaSobre	alfanumérico	true	Parámetro OBLIGATORIO **SÓLO** cuando es un botón de la **VENTANA DE SELECCIÓN**. Indica los campos donde se volcarán los valores elegidos en la ventana de selección.	

Ejemplos de uso del plugin CWBoton:

Ejemplo: Declaración de CWBoton.

```
{CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
```

Lista de plugins [168]

B.1.7. CWBotonTooltip

Estos componentes se utlizan para comunicar la capa de presentación con la capa de la lógica de negocio. Las acciones realizadas a través de ellos no persisten hasta que se confirman con los botones inferiores.

- **Plugins que pueden contener a CWBotonTooltip**
  - Padres
    - CWBarraSupPanel
    - CWFicha
    - CWFila
    - CWSelector
    - CWSolapa
- **Plugins que puede contener CWBotonTooltip**
  - Hijos
    - El plugin CWBotonTooltip es una hoja (no contiene otros plugins)

Tabla de argumentos de CWBotonTooltip

Nombre	Tipo	¿Opcional?	Descripción	Valores
titulo	alfanumérico	false	Nombre para identificar la instancia del componente. Este debe coincidir con la acción que se quiera realizar.	

Nombre	Tipo	¿Opcional?	Descripción	Valores
id	alfanumérico	true	Nombre que identificar el botón. Debe coincidir con la acción que se quiera realizar.	
actuaSobre	alfanumérico	false	En el caso de la actualización de campos aqui se indicará el array de los campos a actualizar. En el resto de casos, indica sobre que panel se va a mostrar el resultado.	Indicará el destino de la acción del botón: • *tabla*: si los datos se muestran sobre una tabla y las operaciones se realizan sobre ella • *ficha*: si los datos se muestran sobre una ficha y las operaciones se realizan sobre ella
imagen	alfanumérico	false	Nombre del fichero que contiene la imagen que queremos que aparezca en el botón, pero sin añadirle la extensión.	
action	alfanumérico	true	Se utiliza para redireccionar en el mapping, la lógica de negocio. **IMPORTANTE:** Botón insertar en panel de búsqueda action siempre valdrá "nuevo".	
funcion	enumerado	true	Indica la función que va a tener asociada ese BotonTooltip (es decir, el tipo de botón que es)	Acción que realizará el botón: • *insertar*: Un botonTooltip de inserción o nuevo • *modificar*: BotonTooltip de edición o modificación • *eliminar*: BotonToltip de marcado para borrar • *limpiar*: BotonToltip de restablecimiento de valores • *abrirVS*: Botón que abre una Ventana de selección • *buscarVS*: Botón que ejecuta la búsqueda dentro de la ventana de selección • *actualizaCampos*: Botón que actualizará una serie de campos

Nombre	Tipo	¿Opcional?	Descripción	Valores
				• *ayuda*: Botón que nos abrirá el manual de ayuda en una ventana emergente.
panelActua	alfanumérico	true	Parámetro **OBLIGATORIO** en el caso de **VENTANA DE SELECCIÓN**, donde será el nombre del panel sobre el que devolverá los datos. También será OBLIGATORIO cuando sea un botón con función 'acutalizaCampos' donde tendrá el valor del parámetro 'id' del panel en el que está.	
filaActual	alfanumérico	true	Parámetro **SÓLO** para el botón buscar de la **VENTANA DE SELECCIÓN** que será una variable smarty de gestión interna.	
formActua	alfanumérico	true	Parámetro OBLIGATORIO **SÓLO** cuando es un botón de la **VENTANA DE SELECCIÓN**. Indica el nombre del formulario en el que se devolverán los valores elegidos en la ventana de selección.	Panel sobre el que actuará: • *fil* • *edi* • *ediDetalle* • *lis* • *lisDetalle*
claseManejadora	alfanumérico	true	Nombre de la clase php que maneje el panel.	
rutaManual	alfanumérico	false	Indicaremos la ruta relativa a partir del directorio doc, a la página del manual que queremos que se muestre. Obligatorio si el botón es un enlace al manual.	

Ejemplos de uso del plugin CWBotonTooltip:

Ejemplo: Botón de eliminar que se encuentra en un panel tabular e insertará en él mismo.

```
{CWBotonTooltip imagen="01" titulo="Eliminar registros" funcion="eliminar" actuaSobre="tabla"}
```

Ejemplo: Botón para inserción en un patrón Tabular-Registro, la inserción se efectuará en el panel registro (ficha) y hay que indicarle un action, con "nuevo" que será la redirección indicada en el mappings.php.

```
{CWBotonTooltip imagen="01" titulo="Insertar registros" funcion="insertar" actuaSobre="ficha" action="nuevo"}

$this->_AddMapping('claseManejadora__nuevo', 'claseManejadora');
$this->_AddForward('claseManejadora__nuevo', 'gvHidraSuccess', 'index.php?view=views/patronesSimples/p_tabularRegistro.php&panel=editar');
$this->_AddForward('claseManejadora__nuevo', 'gvHidraError', 'index.php?view=views/patronesSimples/p_tabularRegistro.php&panel=listar');
```

Ejemplo: Botón asociado a una ventana de selección.

```
{CWBotonTooltip imagen="13" titulo="solucionar" formActua="edi" panelActua="FichaEdicion"
```

```
actuaSobre='autor,fsolucion,l_estado' funcion="actualizaCampos"}
```

Lista de plugins [168]

B.1.8. CWCampoTexto

Equivalente al INPUT de HTML.

- **Plugins que pueden contener a CWCampoTexto**
  - Padres
    - CWContenedor
    - CWFila
    - CWFicha
    - CWSelector
    - CWSolapa
- **Plugins que puede contener CWCampoTexto**
  - Hijos
    - El plugin CWCampoTexto es una hoja (no contiene otros plugins)

Tabla de argumentos de CWCampoTexto

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
datatype	matriz	false	Matriz con una estructura definida en la clase del panel, que definirá el tipo de dato a mostrar en el campo (cadena, número, fecha...) y sus propiedades como: longitud, obligatorio, máscara, calendario... Será una variable smarty en la tpl, definida de la siguiente forma: dataType = $dataType_ClaseManejadora.NombreCampo
longitudMinima	entero	true	Indica el número de carácteres mínimos que debe introducirse en el campo. Si no se alcanza el número mínimo de caracteres a introducir, se muestra un mensaje de ALERTA.
longitudMaxima	entero	true	Indica el número de carácteres máximos que pueden introducirse en el campo. Si se sobrepasa dicho valor, se muestra un mensaje de ALERTA.
editable	enumerado	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWCampoTexto sea hijo

Nombre	Tipo	¿Opcional?	Descripción
			de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
oculto	boolean	true	Añadido a partir de la versión 1.20. Se utiliza para crear CWCampoTextos no visibles, pero que pueden ser útiles para campos calculados, etc... Su comportamiento es similar al de un CWCampoTexto, a escepcion de la parte visual (CSSs) y el javascript de control.
size	alfanumérico	true	Permite indicar el tamaño del campo a la hora de visualizarlo en pantalla. En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.
conUrl	boolean	true	Si aparece y su valor es true, al lado de la caja de texto, aparece un botón, que al pulsarse abrirá una nueva ventana del navegador, cargando la URL que tenga como valor el CampoTexto. Deberá ser una URL válida y completa.
value	alfanumérico	true	Valor que queremos que tenga el campo cuando estamos en una inserción.
actualizaA	alfanumérico	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Fija el texto descriptivo que acompaña a un campo (etiqueta). Si además aparece el argumento/parámetro "obligatorio" a true, se le añade un * que indicará que el campo es obligatorio de rellenar. El texto que se incorpora como etiqueta, será también el que se utilice en los mensajes de comprobación de cmapos obligatorios, etc...
mostrarTextoAsociado	boolean	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
funcion	alfanumérico	false	Llamada a distintas funciones con sus <COMPLETAR>
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.

Ejemplos de uso del plugin CWCampoTexto:

Ejemplo de uso del atributo máscara:

```
{CWCampoTexto nombre="ediCif" size="9" editable="true" textoAsociado="CIF" dataType=$smty_dataType_Personas.ediCif}
```

Lista de plugins [168]

B.1.9. CWContendorPestanyas

Plugin que alberga dentro las pestañas asociadas a un panel.

- **Plugins que pueden contener a CWContendorPestanyas**
  - Padres
    - CWMarcoPanel
- **Plugins que puede contener CWContendorPestanyas**
  - Hijos
    - CWPestanya

Tabla de argumentos de CWContendorPestanyas

Nombre	Tipo	¿Opcional?	Descripción
id	alfanumérico	false	Establece el identificador del CWContenedorPestanyas, y el nombre del objeto JS que controla las pestañas

Ejemplos de uso del plugin CWContendorPestanyas:

Ejemplo: Declaración de un CWContenedorPestanya con dos pestanyas dentro.

```
{CWContenedorPestanyas id="Maestro"}
  {CWPestanya tipo="fil" estado=$estado_fil}
  {CWPestanya tipo="lis" estado=$estado_lis}
{/CWContenedorPestanyas}
```

Lista de plugins [168]

B.1.10. CWContenedor

Componente sin funcionalidad visual, se utiliza para albergar tanto tablas de HTML como plugins, desde componentes básicos hasta plugins fichas y/o tablas.

- **Plugins que pueden contener a CWContenedor**
  - Padres
    - CWPanel
- **Plugins que puede contener CWContenedor**
  - Hijos
    - CWTabla
    - CWFichaEdicion
    - CWFicha (sólo en dos casos: Una búsqueda o un campo external en un tabular)

Tabla de argumentos de CWContenedor

El plugin CWContenedor No tiene argumentos

Ejemplos de uso del plugin CWContenedor:

Ejemplo: Contenedor con el componente CWTabla.

```
{CWContenedor}
  {CWTabla ...}
   ...
  {/CWTabla}
{/CWContenedor}
```

Ejemplo: Contenedor con el componente CWFichaEdicion.

```
{CWContenedor}
  {CWFichaEdicion ...}
   ...
  {/CWFichaEdicion}
{/CWContenedor}
```

Ejemplo: Contenedor para panel de búsqueda.

```
{CWContenedor}
  {CWFicha}
    <br/>Busqueda por fechas<br/>
    {CWCampoTexto nombre="filFechaIni" size="10" editable="true" textoAsociado="Fecha recepcion inicial:"}
    {CWCampoTexto nombre="filFechaFin" size="10" editable="true" textoAsociado="Fecha recepcion final:"}
  {/CWFicha}
{/CWContenedor}
```

Lista de plugins [168]

B.1.11. CWCheckBox

Equivalente al CHECKBOX de HTML.

- **Plugins que pueden contener a CWCheckBox**
  - Padres
    - CWContenedor
    - CWFila
    - CWFicha
    - CWSolapa
    - CWSelector
- **Plugins que puede contener CWCheckBox**
  - Hijos
    - El plugin CWCheckBox es una hoja (no contiene otros plugins)

Tabla de argumentos de CWCheckBox

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
editable	enumerado	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en

Nombre	Tipo	¿Opcional?	Descripción
			la inserción cuando el plugin CWCheckBox sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
valor	alfanumérico	true	Valor que se le pasa a la capa de negocio cuando el componente este en un panel de búsqueda. Será obligatorio siempre que forme parte de un panel de búsqueda.
datatype	matriz	false	Matriz con una estructura definida en la clase del panel, que definirá las propiedades del campo: obligatorio, valor chequeado, valor no chequeado. Será una variable smarty en la tpl, definida de la siguiente forma: dataType = $dataType_ClaseManejadora.NombreCampo
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Fija el texto descriptivo que acompaña a un campo (etiqueta). Si además aparece el argumento/parámetro "obligatorio" a true, se le añade un * que indicará que el campo es obligatorio de rellenar. El texto que se incorpora como etiqueta, será también el que se utilice en los mensajes de comprobación de cmapos obligatorios, etc...
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.
size	alfanumérico	true	En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.

Ejemplos de uso del plugin CWCheckBox:

Declaración de CWCheckBox como campo de una tabla.

```
{CWCheckBox nombre="descEstado" editable="true" dataType=$dataType_claseManejadora.descEstado}
```

SUBIR [168]

B.1.12. CWFicha

El plugin ficha, se encarga de dibujar las capas HTML y el campo oculto que indica en que estado se encuentra la ficha (modificación, inserción o borrado). En el caso de los paneles de búsqueda, como en el uso de campos "external" (campos comunes a todas las tuplas de un panel tabular) este plugin tendrá como padre al CWContenedor directamente.

- **Plugins que pueden contener a CWFicha**

- Padres
  - CWFichaEdicion
  - CWContenedor

- **Plugins que puede contener CWFicha**
  - Hijos
    - CWCampoTexto
    - CWAreaTexto
    - CWLista
    - CWCheckBox
    - CWSelector
    - CWSolapa

Tabla de argumentos de CWFicha

El plugin CWFicha no tiene argumentos

Ejemplos de uso del plugin CWFicha:

Uso del plugin CWFicha en un panel de edición.

```
{CWFichaEdicion id="FichaEdicion" datos=$smty_datosTablaM}
  {CWFicha}
    {CWCampoTexto nombre="anyo" editable="true" size="4" value=$smty_anyoNuevo textoAsociado="A&ntilde;o"}
      
    {CWCampoTexto nombre="nfactura" editable="false" size="6" value="0" textoAsociado="N&uacute;mero Factura"}
    <br><br>
    {CWLista nombre="procedencia" radio="true" editable="true" datos=$smty_datosPreInsertadosTinvEntradas2.procedencia textoAsociado="Procedencia"}
  {/CWFicha}
  {CWPaginador enlacesVisibles="3"}
{/CWFichaEdicion}
```

Lista de plugins [168]

B.1.13. CWFichaEdicion

Plugin que alberga dentro un CWFicha, a partir de los datos que le lleguen como parámetros y de su contenido, representa esos datos como fichas, si se desea poder moverse entre ellas, debe incluirse un CWPaginador.

- **Plugins que pueden contener a CWFichaEdicion**
  - Padres
    - CWContenedor

- **Plugins que puede contener CWFichaEdicion**
  - Hijos
    - CWFicha
    - CWPaginador

Tabla de argumentos de CWFichaEdicion

Nombre	Tipo	¿Opcional?	Descripción
id	alfanumérico	false	Establace el identificador del CWFichaEdicion
datos	array asociativo	false	Array asociativo con los datos que queremos mostrar en la tabla. Los componentes que representen los datos de este array deben llamarse igual que las claves del array. Será una variable smarty ($smty_datos) que la sustituye IgepPanel.

Ejemplos de uso del plugin CWFichaEdicion:

```
{CWFichaEdicion id="FichaEdicion" datos=$smty_datosFicha}
  {CWFicha}
    ...
  {/CWFicha}
{/CWFichaEdicion
```

Lista de plugins [168]

B.1.14. CWFila

Este plugin representa la fila de una tabla. Se utiliza como molde para manejar la información en modo browse.

- **Plugins que pueden contener a CWFila**
  - Padres
    - CWTabla
- **Plugins que puede contener CWFila**
  - Hijos
    - CWCampoTexto
    - CWAreaTexto
    - CWLista
    - CWCheckBox

Tabla de argumentos de CWFila

Nombre	Tipo	¿Opcional?	Descripción
tipoListado	booleano	true	Se utiliza para las tablas que queramos que unicamente muestre la informacion en plan listado, no utilizaremos componentes básicos para representar los datos.

Ejemplos de uso del plugin CWFila:

```
{CWFila tipoListado="false"}
  {CWCampoTexto nombre="lisCif" size="13" textoAsociado="CIF" dataType=$dataType_Registro.lisCif}
  {CWCampoTexto nombre="lisOrden" size="2" textoAsociado="Orden" dataType=$dataType_Registro.lisOrden}
  {CWLista nombre="lisDepartamento" textoAsociado="Departamento" datos=$defaultData_Registro.lisDepartamento dataType=$dataType_Registro.lisOrden}
{/CWFila}
```

Lista de plugins [168]

B.1.15. CWImagen

Equivalente al FILE de HTML.

- **Plugins que pueden contener a CWImagen**
  - Padres
    - CWFicha
- **Plugins que puede contener CWImagen**
  - Hijos
    - El plugin CWImagen es una hoja (no contiene otros plugins)

Tabla de argumentos de CWImagen

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
width	entero	true	Indica el ancho de la imagen.
height	entero	true	Indica el alto de la imagen.
alt	enumerado	true	Texto alternativo que se representa cuando la imagen no puede ser mostrada.
src	entero	true	Ruta a la imagen. Cuando se ha de mostrar una imagen de la bd no es obligatorio poner este parámetro.
rutaAbs	booleano	true	Ruta absoluta. Indica que debe buscar la imagen en la ruta absoluta del servidor.
textoAsociado	alfanumérico	false	Texto que acompañará a la imagen.
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
visible	booleano	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.

Ejemplos de uso del plugin CWImagen:

```
{CWImagen nombre="fichero" src="imagenes/imagen0.gif" rutaAbs="true"}
```

Lista de plugins [168]

B.1.16. CWLista

Es un componente de seleccion, simple y/o multiple, se corresponde con los elementos de selección HTML, es decir, las etiquetas SELECT (select-one y select-multiple) y RADIOBUTTON. A través de los argumentos del plugin pueden expresarse selecciones condicionales, por ejemplo, listas dependientes unas de otras (Ej. Provincia - Municipio).

- **Plugins que pueden contener a CWLista**
  - Padres
    - CWFila
    - CWFicha
- **Plugins que puede contener CWLista**
  - Hijos
    - El plugin CWLista es una hoja (no contiene otros plugins)

Tabla de argumentos de CWLista

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
editable	enumerado	true	Especifica el comportamiento del selector: si su valor es "true", es editable por el usuario. Si el valor es "false", no es editable y si su valor es "nuevo", será editable solo en la inserción. Si no se especifica el atributo el campo es editable.
datatype	matriz	false	Matriz con una estructura definida en la clase del panel, que definirá las propiedades de la lista, tales como obligatoriedad, tamaño (size), si es una lista múltiple o no, y si queremos que aparezca como un elemento tipo RadioButton o tipo Select. Será una variable smarty en la tpl, definida de la siguiente forma: dataType = $dataType_ClaseManejadora.NombreCampo
numCaracteres	entero	true	Indica el número de caracteres que se mostrarán en la lista desplegable (ancho). En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.
obligatorio	booleano	true	Especifica el comportamiento del componente: si el su valor es 'true', es necesario que el campo tenga valor, o mostrará un mensaje de ALERTA. Si su valor es 'false', no es necesario rellenarlo. cuando el plugin CWLista sea hijo de CWFila o CWFicha. Si no se especifica el atributo, la introducción de valores en el campo no es obligatoria.

Nombre	Tipo	¿Opcional?	Descripción
multiple	booleano	true	Si se especifica la lista de selección se convierte en una lista de selección multiple.
size	entero	true	Numero máximo de opciones del desplegable sin que aparezca el scroll.
radio	booleano	true	Si aparece este parámetro, el plugin CWLista, genera un grupo de radio Buttoms con las opciones indicadas. No es compatible con el argumento "multiple".
datos	matriz	false	Matriz con una estrutura definida, que se le pasa al CWLista para indicar cuales son las opciones posibles, el valor de cada una y cual es la seleccionada. Por lo tanto es una variable smarty ($smty_datosPreinsertados[claseManejadora]) que será sustituida por IgepPanel. Si el componente se encuentra dentro de un CWTabla o CWFicha, recogerá esos datos del padre (es decir, del CWTabla o del CWFicha). En ese caso éste argumento se utiliza para indicar cuales son los valores que deben mostrarse cuando se inserta un nuevo registro.
actualizaA	alfanumérico	true	Se utiliza este campo en el caso de listas dependientes. Se pondra el nombre del campo que actualizas cuando este toma un valor.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Texto que acompaña a un campo. Si además aparece el argumento obligatorio a true, se le añade un * que indicará que el campo es obligatorio de rellenar.
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
visible	booleano	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.

Ejemplos de uso del plugin CWLista:

Ejemplo: Dos listas dependientes.

```
{CWLista nombre="ediCodProv" radio="true" size="3" actualizaA="ediCodMun" editable="true" datos=$defaultData_Registro.ediCodProv dataType=$dataType_Registro.ediCodProv}
{CWLista nombre="ediCodMun" size="3" editable="true" dataType=$dataType_Registro.ediCodMun datos=$defaultData_Registro.ediCodMun}
```

Lista de plugins [168]

B.1.17. CWMarcoPanel

Se utiliza para agrupar paneles, cada panel representa un modo de trabajo (busqueda, ficha, browse...) sobre un mismo conjunto de datos o un subconjunto de los mismos. Sus hijos serán siempre uno o más paneles y por último, un CWContenedorPestanyas si es que estas deben aparecer.

- **Plugins que pueden contener a CWMarcoPanel**
  - Padres
    - CWVentana
- **Plugins que puede contener CWMarcoPanel**
  - Hijos
    - CWPanel
    - CWContenedorPestanyas

Tabla de argumentos de CWMarcoPanel

Nombre	Tipo	¿Opcional?	Descripción
conPestanyas	booleano	true	Debe utilizarse en cuanto aparezcan los modos de comportamiento. Cuando su valor es true, introduce los elementos Javascript necesarios para el manejo de las pestañas.

Ejemplos de uso del plugin campo:

Ejemplo: Uso del componente CWMarcoPanel en un Maestro-Detalle.

```
{CWVentana ...}
...
 {CWMarcoPanel conPestanyas="true"}

<!--*********** PANEL fil ******************-->
   {CWPanel id="fil" action="buscar" method="post" estado="$estado_fil" claseManejadora="Registro"}
   ...
   {/CWPanel}
<!--*********** PANEL lis ******************-->
   {CWPanel id="edi" tipoComprobacion="envio" action="operarBD" method="post" estado="$estado_edi" claseManejadora="Registro"}
   ...
   {/CWPanel}
<!-- ***************** PESTANYAS ************************-->
   {CWContenedorPestanyas}
     {CWPestanya tipo="fil" estado=$estado_fil}
     {CWPestanya tipo="edi" estado=$estado_edi}
   {/CWContenedorPestanyas}
 {/CWMarcoPanel}
{/CWVentana}
```

Lista de plugins [168]

B.1.18. CWMenuLayer

Adaptación simplificada del menu dinámico generado en PHP y JavaScript del proyecto GPL PHPLM (PHP Layers Menu) de Marco Pratesi

- **Plugins que pueden contener a CWMenuLayer**
  - Padres
    - CWBarra
- **Plugins que puede contener CWMenuLayer**
  - Hijos
    - El plugin CWMenuLayer es una hoja (no contiene otros plugins)

Tabla de argumentos de CWMenuLayer

Nombre	Tipo	¿Opcional?	Descripción
name	alfanumérico	false	Especifica el nombre del componente. Si no se indica, se genera uno automáticamente. Es una variable smarty ($smty_nombre) que se sustituirá en IgepPanel.
usarImagenesAplicacion	booleano	false	Parámetro opcional, cuyo valor por defecto es "false", indica si debe buscar las imágenes del menu en la aplicación, o en caso de ser falso, utilizará las imágenes de gvHidra.
imgDescenso	alfanumérico	true	Indica el nombre de la imagen (soporta gif, jpg o png) que despliega el menu de forma descedente. La imagen por defecto es down-arrow.png
imgDespliega	alfanumérico	true	Indica el nombre de la imagen (soporta gif, jpg o png) que despliega el menu de forma lateral. La imagen por defecto es forward-arrow.png
fichero	alfanumérico	true	Parámetro opcional con el que se indica el fichero (extensión .str) que contiene la estructura del menu el formato del mismo es: ". | opcion | url | texto ayuda emergente | imagen" Si se utiliza esta forma de establecer el menú no ha de existir el próximo parámetro 'cadenaMenu'.
cadenaMenu	alfanumérico	true	Cadena de texto con la estructura del menú. Al ser una cadena de texto, podemos generar menús dinámicos (será un fichero xml), es decir, cuyas opciones varíen en función de algún parámetro, por ejemplo en función del ROL del usuario etc... Será una variable smarty $smty_cadenaMenu que se sustituirá en IgepPantalla.

Ejemplos de uso del plugin CWMenuLayer:

Ejemplo de la estuctura del menú

```
.|Menu|
..|Entrada Opción 1||
...|Inserción|?view=views/insercion.php| Opción para Nuevo Registro | insertar.png
...|Mantenimiento|?view=views/mantenimiento.php| Opción para Mantenimiento
..|Entrada Opción 2||
...|Prueba con frame oculto|?view=views/inicio.php| Aplicación
...|Prueba Maestro-Detalle|?view=views/MD.php| Home en flaco
```

Ejemplo: Dentro de CWVentana.

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
  {CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=$smty_customTitle}
    {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
  {/CWBarra}
  {CWMarcoPanel conPestanyas="true"}
   ...
  {/CWMarcoPanel}
{/CWVentana}
```

Lista de plugins [168]

B.1.19. CWPaginador

Integra una línea de enlaces para paginar cuando se presentan múltiples regitros a través de los plugins CWTabla y CWFicha.

- **Plugins que pueden contener a CWPaginador**
  - Padres
    - CWTabla
    - CWFichaEdicion
- **Plugins que puede contener CWPaginador**
  - Hijos
    - El plugin CWPaginador es una hoja (no contiene otros plugins)

Tabla de argumentos de CWPaginador

Nombre	Tipo	¿Opcional?	Descripción
pagInicial	entero	true	Indica la página que aparecerá visible cuando se cargue la pantalla, por defecto siempre vale 0.
enlacesVisibles	entero	true	Indica el número de enlaces que apareceran para realizar la navegación además de los fijos (siguiente, ultimo, anterior y primero)

Ejemplos de uso del plugin CWPaginador:

Ejemplo: Utilizacion de un CWPaginador en una tabla.

```
{CWTabla conCheck="true" seleccionUnica="true" id="Tabla1" datos=$smty_datosTabla}
  {CWFila tipoListado="false"}
    {CWCampoTexto nombre="lisCif" size="9" editable="true" textoAsociado="CIF" dataType=$smty_dataType_Personas.lisCif}
    {CWCampoTexto nombre="lisNombre" editable="true" size="30" textoAsociado="Nombre" dataType=$smty_dataType_Personas.lisNombre}
  {/CWFila}
  {CWPaginador enlacesVisibles="3"}
{/CWTabla}
```

Lista de plugins [168]

B.1.20. CWPanel

Este componente es un contenedor, aporta javascript, una tabla HTML para incluir otros plugins, y un elemento FORM de HTML, por lo que es el plugin que realiza los envios de información hacia la lógica. Cada panel es un formulario HTML y se corresponde con una de las pestañas de los modos de trabajo indicados en la guía de estilo (búsqueda, ficha, tabular).

El orden de los hijos en un panel si que es importante y debe respetarse.

- **Plugins que pueden contener a CWPanel**
  - Padres
    - CWMarcoPanel
- **Plugins que puede contener CWPanel**
  - Hijos (hay que respetar este orden a la hora de crear los hijos)

- CWBarraSupPanel
- CWContenedor
- CWBarraInfPanel

Tabla de argumentos de CWPanel

Nombre	Tipo	¿Opcional?	Descripción	Valores
id	enumerado	false	Fija el identificador del componete y del formulario HTML que incluye por debajo. Es necesario que dicho nombre sea único para evitar comportamientos no previstos en la interfaz.	Puede tomar los siguientes valores: • *fil*: panel de búsqueda. • *edi*: panel registro. • *lis*: panel tabulares. • *ediMaestro / ediDetalle*: panel registro maestro o detalle, respectivamente. • *lisMaestro / lisDetalle*: panel tabular maestro o detalle, respectivamente.
action	alfanumérico	true	Tipo de acción que debe coincidir con la correspondiente en el mappings.	
accion	alfanumérico	true	Estado en el que se quiere encontrar el panel al cargarlo (modo inserción, modificación, borrado)	Puede tomar los siguientes valores: • *insertar* • *modificar* • *borrar*
tipoComprobacion	alfanumérico	false	Indica la comprobación que queremos que se realice a los campos que tengan el argumento comprobacion igual true. Por defecto hará una comprobación "envio"	Puede tomar los siguientes valores: • *envio* • *foco* • *todo*
method	alfanumérico	false	Indica la manera en que debe realizarse el submit del FORM HTML (get/post).	
estado	alfanumérico	true	Indica el estado del panel (activo o desactivado) cuando se carga la pantalla.	Puede tomar los siguientes valores:

Nombre	Tipo	¿Opcional?	Descripción	Valores
			Las opciones son, si no viene dado desde la capa de negocio con variables smarty:	• *on*: visible y activo • *off*: visible e inactivo • *inactivo*: no visible e inactivo
claseManejadora	alfanumérico	true	Indica la clase que va a ocuparse de la lógica de esta pantalla.	

Ejemplos de uso del plugin CWPanel:

Ejemplo de uso del action:

```
-> Parámetro del plugin
{CWPanel id="edi" action="operarBD" estado="$estado_edi" claseManejadora="TinvEntradas2" ...}

-> Correspondencia en el mappings.php
$this->_AddMapping('TinvEntradas2__operarBD', 'TinvEntradas2', '', 'IgepForm', 0);
```

Ejemplo: Uso del componente CWPanel en un panel de búsqueda.

```
{CWPanel id="fil" action="buscar" method="post" estado="$estado_fil" claseManejadora="TinvDonantes"}
  {CWBarraSupPanel titulo="Donantes"}
    {CWBotonTooltip imagen="01" titulo="Insertar registro" funcion="insertar" actuaSobre="ficha" action="nuevo"}
    {CWBotonTooltip imagen="04" titulo="Limpiar campos" funcion="limpiar" actuaSobre="ficha"}
  {/CWBarraSupPanel}
  {CWContenedor}
    {CWFicha}
      <br>  
      {CWCampoTexto nombre="filCif" editable="true" size="13" textoAsociado="CIF/NIF" dataType=$dataType_TinvDonantes.filCif}
      <br><br>
    {/CWFicha}
   {/CWContenedor}
   {CWBarraInfPanel}
     {CWBoton imagen="50" texto="Buscar" class="boton" accion="buscar"}
   {/CWBarraInfPanel}
{/CWPanel}
```

Lista de plugins [168]

B.1.21. CWPantallaEntrada

Con este plugin creamos la pantalla inicial de cualquier aplicación según la guía de estilo. Todos sus parámetros son variables smarty que se sustituyen internamente.

- **Plugins que pueden contener a CWPantallaEntrada**
  - Padres
    - CWVentana
- **Plugins que puede contener CWPantallaEntrada**
  - Hijos
    - El plugin CWPantallaEntrada es una hoja (no contiene otros plugins)

Tabla de argumentos de CWPantallaEntrada

Nombre	Tipo	¿Opcional?	Descripción
usuario	alfanumérico	false	Se utiliza para mostrar en pantalla el usuario que está conectado a la aplicación.

Nombre	Tipo	¿Opcional?	Descripción
nomApl	alfanumérico	false	Se utiliza para mostrar el nombre de la aplicación que se va a utilizar.
codApl	alfanumérico	false	Se utiliza para mostrar en pantalla el código de la aplicación. (Abreviatura que suele corresponderse con el nombre del directorio del servidor Web donde se ubica)
rolApl	alfanumérico	false	Pendiente de concretar. Roles de la aplicación

Ejemplos de uso del plugin CWPantallaEntrada:

Ejemplo: Declaración de una Pantalla de inicio. Sólo aparece en aplicación.tpl, plantilla exlclusiva de igep.

```
{CWVentana tipoAviso=$smty_tipoAviso titulo=$smty_tituloApl codAviso=$smty_codError descBreve=$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad onUnload=
$smty_jsOnUnload}
  {CWPantallaEntrada usuario=$smty_usuario rolApl=$smty_rolApl nomApl=$smty_aplicacion codApl=$smty_codaplic}
{/CWVentana}
```

Lista de plugins [168]

B.1.22. CWPestanyas

Plugin que dibujará la pestaña lateral correspondiente.

- **Plugins que pueden contener a CWPestanyas**
  - Padres
    - CWContenedorPestanyas
- **Plugins que puede contener CWPestanyas**
  - Hijos
    - El plugin CWPestanyas es una hoja (no contiene otros plugins)

Tabla de argumentos de CWPestanyas

Nombre	Tipo	¿Opcional?	Descripción	Valores
tipo	enumerado	false	Indicará el tipo de pestaña	El valor a tomar dependerá del panel al que va asociado: • *fil*: Panel filtro • *edi*: Panel registro • *lis*: Panel tabular
estado	alfanumérico	false	Valor que corresponderá con el estado que queramos que aparezaca en un principio. Puede ser establecida por defecto en la tpl o ser una variable de smarty sustituida internamente.	
panelAsociado	alfanumérico	false	Nombre del panel al que está asociada dicha pestaña.	

Nombre	Tipo	¿Opcional?	Descripción	Valores
ocultar	alfanumérico	false	Se utiliza en los mantenimientos mestro detalle, nos permite ocultar el panel de detalle cuano se se selecciona la pestaña en el que está definido el parámetro (el valor es "Detalle")	
mostrar	alfanumérico	false	Similar a ocultar, pero en este caso se utiliza para activar el detalle ('Detalle' como argumento) al seleccionar la pestaña que tiene dicho parámetro	

Ejemplos de uso del plugin CWPestanyas:

Ejemplo: Definición de dos pestañas.

```
{CWContenedorPestanyas}
  {CWPestanya tipo="fil" estado=$estado_fil}
  {CWPestanya tipo="lis" estado=$estado_lis}
{/CWContenedorPestanyas}
```

Lista de plugins [168]

B.1.23. CWSelector

Define un patrón tipo lista que nos permita simular el maestro-detalle-subdetalle. Por un lado tenemos una lista múltiple donde se irán acumulando los valores procedentes de los campos que se hayan incluido en el selector. Estos campos pueden ser un conjunto de elementos básicos (CWCampoTexto, CWLista (NO multiple), CWCheckBox, CWBotonToolTip) o una única lista multiple. No se podrá combinar una lista múltiple con otros elementos de formulario (campos de texto, listas simples...), ya que las opciones elegidas de la lista múltiple se copiarán a la lista destino como opciones diferentes y no una combinación como la que se hace con varios campos.

- **Plugins que pueden contener a CWSelector**
  - Padres
    - CWFicha
- **Plugins que puede contener CWSelector**
  - Hijos
    - CWCampoTexto
    - CWAreaTexto
    - CWLista
    - CWCheckBox
    - CWBotonToolTip

Tabla de argumentos de CWSelector

Nombre	Tipo	¿Opcional?	Descripción
titulo	alfanumérico	false	Título que identifica el elemento.

Nombre	Tipo	¿Opcional?	Descripción
botones	matriz	false	Estructura recibida de la capa de negocio para indicar que botones ('insertar','modificar','eliminar') aparecerán visibles.
nombre	alfanumérico	false	Nombre para el campo destino donde se irán acumulando los valores.
editable	alfanumérico	true	Especifica el comportamiento del campo destino: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWAreaTexto sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
separador	alfanumérico	true	Carácter que utilizaremos para separar los distintos datos que formen parte de un mismo registro, por defecto será el carácter "|".
datos	matriz	true	Matriz con una estrutura definida, que se le pasa al CWSelector para mostrar los datos existentes a los que podremos añadir o eliminar otros. Será una variable smarty $smty_datos que se encargará IgepPanel de sustituir.
rows	entero	true	Especifica el número de filas que tendrá el textarea.

Ejemplos de uso del plugin CWSelector:

Ejemplo: Selector con varios campos de texto.

```
{CWSelector titulo="*Formato:" botones=$smty_botones nombre="listaBienes" editable="nuevo" separador=","}
    {CWCampoTexto nombre="ediCcentro" editable="nuevo" size="2" textoAsociado="Centro" actualizaA="ediDcentro" dataType=$dataType_MiClase.ediCcentro}
    &amp;nbsp;&amp;nbsp;
    {CWCampoTexto nombre="ediDcentro" editable="false" size="50" dataType=$dataType_MiClase.ediDcentro}
    {CWBotonTooltip imagen="13" titulo="Ventana de seleccion" funcion="abrirVS" actuaSobre="ccentro" formActua="ediDetalle" panelActua="FichaDetalle" claseManejadora="TinvLineas2"}
    <br><br>
    {CWCampoTexto nombre="ediUds" editable="true" size="5" textoAsociado="Uds" dataType=$dataType_MiClase.ediUds}
    <br><br>
{/CWSelector}
```

Ejemplo: Selector con una lista con datos de la BD.

```
{CWSelector titulo="*Formato:" botones=$smty_botones nombre="listaBienes" editable="nuevo" datos=$defaultData_Registro.listaBienes}
    {CWLista multiple=true nombre="categorias" editable="nuevo" textoAsociado="Categorias" datos=$defaultData_Registro.categorias}
{/CWSelector}
```

Lista de plugins [168]

B.1.24. CWSolapa

Plugin que alberga dentro las solapas asociadas a un panel. Genera la lógica JavaScript necesaria para manejarlas. Es necesario que las solapas estén dentro de una FichaEdicion, que tiene como parámetros 'numSolapas' y 'titulosSolapas' (ver plugin CWFichaEdicion).

- **Plugins que pueden contener a CWSolapa**
  - Padres
    - CWFicha
- **Plugins que puede contener CWSolapa**
  - Hijos

- CWCampoTexto
- CWAreaTexto
- CWLista
- CWCheckBox

Tabla de argumentos de CWSolapa

Nombre	Tipo	¿Opcional?	Descripción
titulo	alfanumérico	false	Establace el texto que aparecerá en la solapa.
poscionSolapa	entero	false	Es un atributo obligatorio, que indica la posicion que tendra esa solapa. Debe ser consecutivo y empezar por 0.

Ejemplos de uso del plugin CWSolapa:

Ejemplo: Declaración de un CWContenedorPestanya con dos pestanyas dentro.

```
{CWFichaEdicion id="FichaEdicion" datos=$smty_datosFicha}
  {CWFicha}
    {CWSolapa titulo=" Datos 1" posicionSolapa=0}
      {CWCampoTexto nombre="ediCif" editable="true" size="13" textoAsociado="CIF" dataType=$dataType_Registro.ediCif}
      {CWCampoTexto nombre="ediOrden" editable="true" size="2" textoAsociado="Orden" dataType=$dataType_Registro.ediOrden}
    {/CWSolapa}
    {CWSolapa titulo="Datos personales" posicionSolapa="1"}
      ...
    {/CWSolapa}
{/CWFicha}
{/CWFichaEdicion}
```

Lista de plugins [168]

B.1.25. CWTabla

Equivalente al TABLE del HTML

- **Plugins que pueden contener a CWTabla**
  - Padres
    - CWContenedor
- **Plugins que puede contener CWTabla**
  - Hijos
    - CWFila

Tabla de argumentos de CWTabla

Nombre	Tipo	¿Opcional?	Descripción
id	alfanumérico	false	Atributo utilizado por plugins herederos o descendientes. Lo utiliza el plugin CWFila para generar los elementos TR con un id.
datos	array asociativo	false	Vector asociativo que el programador pasa a la tabla con los datos que queremos que se muestre en

Nombre	Tipo	¿Opcional?	Descripción
			esta. Variable smarty ($smty_datos) que se sustituye internamente.
conCheck	boleano	true	Si aparece y su valor es true, al principio de cada fila que compone la tabla aparecera un checkbox que utilizaremos para seleccionar la fila o filas sobre las que realizaremos las diferentes acciones.
seleccionUnica	boleano	true	Permite un único elemento seleccionado en la tabla.
conCheckTodos	boleano	true	Si aparece y su valor es true, aparecera un checkbox en la cabecera de la tabla que nos permitira seleccionar y deseleccionar todos los registros.
numFilasPantalla	entero	true	Fija el número de filas de datos que queremos que aparezca por pantalla. En caso de no aparecer por defecto apareceran 6 filas de datos.

Ejemplos de uso del plugin CWTabla:

```
{CWTabla conCheck="true" seleccionUnica="true" id="Tabla1" datos=$smty_datosTabla}
  {CWFila tipoListado="false"}
     {CWCampoTexto nombre="lisCif" size="9" editable="true" textoAsociado="CIF" dataType=$dataType_MiClase.lisCif}
     {CWCampoTexto nombre="lisNombre" editable="true" size="30" textoAsociado="Nombre" dataType=$dataType_MiClase.lisNombre}
     {CWCampoTexto nombre="lisMoto" editable="true" size="30" textoAsociado="Moto" dataType=$dataType_MiClase.lisMoto}
  {/CWFila}
  {CWPaginador enlacesVisibles="3"}
{/CWTabla}
```

Lista de plugins [168]

B.1.26. CWUpLoad

Equivale al FILE de HTML

- **Plugins que pueden contener a CWUpLoad**
  - Padres
    - El plugin CWUpLoad es raiz (no lo contiene ningún otro plugin)
- **Plugins que puede contener CWUpLoad**
  - Hijos
    - CWFicha
    - CWConenedor

Tabla de argumentos de CWUpLoad

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
obligatorio	booleano	true	Especifica el comportamiento del campo: Si el su valor es true, es necesario que el campo tenga valor,

Nombre	Tipo	¿Opcional?	Descripción
			o mostrará un mensaje de ALERTA. Si su valor es false, no es necesario rellenarlo. Cuando el plugin CWCampoTexto sea hijo de CWFila o CWFicha, si no se especifica el atributo, la introducción de valores en el campo no es obligatoria.
size	entero	true	Tamaño de la caja de texto en pantalla.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Texto que acompaña a un campo. Si además aparece el argumento obligatorio a true, se le añade un * que indicará que el campo es obligatorio de rellenar.

Ejemplos de uso del plugin CWUpLoad:

```
{CWUpLoad nombre="ficheroUpload" size="10" textoAsociado="Si quieres cambiar la imagen..."}
```

Lista de plugins [168]

B.1.27. CWVentana

Este componente se basa en la ventana HTML. Es el plugin raíz, con el se incluye la base javascript necesaria para el comportamiento de la interfaz (manejo de capas, errores...). Todos sus parámetros serán fijados internamente mediante variables smarty.

- **Plugins que pueden contener a CWVentana**
  - Padres
    - El plugin CWVentana es raiz (no lo contiene ningún otro plugin)
- **Plugins que puede contener CWVentana**
  - Hijos
    - CWBarra
    - CWMarcoPanel
    - CWPantallaEntrada
    - CWEjecutarScripts

Tabla de argumentos de CWVentana

Nombre	Tipo	¿Opcional?	Descripción	Valores
titulo	alfanumérico	true	Fija el título de la Ventana HTML. Variable smarty $smty_tituloApl.	
tipoAviso	alfanumérico	false	Indica el tipo de aviso según la guia de estilo. Variable smarty $smty_tipoAviso.	Los tipos de aviso pueden ser: • *aviso* • *alerta*

Nombre	Tipo	¿Opcional?	Descripción	Valores
				• *notificación* • *sugerencia*
codAviso	alfanumérico	false	Fija el código de aviso. Variable smarty $smty_codError.	
descBreve	alfanumérico	false	Descripción breve del mensaje del aviso. Variable smarty $smty_descBreve.	
textoAviso	alfanumérico	false	Descripción detallada del aviso. Variable smarty $smty_textoAviso.	
onload	alfanumérico	false	En este parámetro podremos introducir una llamada a una funcion javascript que debe ejecutarse en el evento onLoad de la página	
onUnload	alfanumérico	false	En este parámetro podremos introducir una llamada a una funcion javascript que debe ejecutarse en el evento onUnLoad de la página	

Ejemplos de uso del plugin CWVentana:

```
{CWVentana tipoAviso=$smty_tipoAviso  codAviso=$smty_codError  descBreve = $smty_descBreve  textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
...
{/CWVentana}
```

Lista de plugins [168]